HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Crowd-basierte Luftqualitätsmessungen mit der Complex Event Processing Engine Siddhi auf Raspberry Pi

Andreas Engelhardt

Bachelor-Arbeit im Studiengang „Angewandte Informatik“

20. September 2020

Autor Andreas Engelhardt
1434446
andreas.engelhardt.04@gmail.com

Erstprüferin: Prof. Dr. Ralf Bruns
Abteilung Informatik, Fakultät IV
Hochschule Hannover
ralf.bruns@hs-hannover.de

Zweitprüfer: M. Sc. Jeremias Dötterl
Abteilung Informatik, Fakultät IV
Hochschule Hannover
jeremias.doetterl@hs-hannover.de

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelor-Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 20. September 2020 Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Mit Beginn der Industrialisierung gegen Ende des 18. Jahrhunderts begann gleichzeitig ein bis heute andauernder Prozess der Umweltverschmutzung. Gewässer, Klima und Lebensräume werden seither zunehmend belastet, wodurch die natürliche Umwelt aller Lebewesen gefährdet wird. Aus diesem Grund richtet die Gesellschaft seit längerer Zeit ihren Blick stark auf die Erhaltung der Umwelt. Ein wesentlicher Aspekt ist dabei die Regulierung der Luftverschmutzung, die nämlich schon vor der Industrialisierung durch sogenanntes „Rauchgas“ aufkam. [Wie]

Die Suche nach „sauberer“ Luft ist somit vergebens, da durch den geschichtlichen Hintergrund deutlich wird, dass es diese in ihrer natürlichen Zusammensetzung so nicht mehr gibt und verunreinigte Luft für uns Menschen in der heutigen Zeit somit der „default“-Case ist. Luftverschmutzung beschreibt nämlich „die Abweichung der Luftzusammensetzung von ihren natürlichen Werten durch die Emission potentieller Schadstoffe“. [Luf] Die Luft, die wir Menschen täglich einatmen beinhaltet also nicht nur umweltschädliche, sondern vor allem für unseren Körper gesundheitsschädliche Schadstoffe. Laut der WHO führt die Luftverschmutzung jedes Jahr zu ca. 7 Millionen Todesfällen weltweit, dem hinzu kommen zahlreiche Herz- und Atemwegserkrankungen. [WHO18] Um dem entgegenzuwirken, wurden europaweit zahlreiche Maßnahmen und Grenzwerte zur Überwachung und Regulierung der Luftverschmutzung festgelegt.[UWB13] Relevant dafür sind folgende Schadstoff:

- Feinstaub (PM_{10} und $PM_{2,5}$)
- Ozon (O_3)
- Kohlenmonoxid (CO)
- Stickstoffdioxid (NO_2)
- Schwefeldioxid (SO_2)

Die Überwachung dieser Schadstoffgrenzwerte erfolgt durch sehr große, fest installierte Messstationen, deren Anzahl und Positionierung sich ebenfalls nach gesetzlichen Vorgaben richtet. Unter anderem sollen sie an Orten mit der stärksten Belastung Luftdaten sammeln, woraus dann die lokale Ausbreitung der Schadstoffe in einem Ballungsgebiet taxiert wird. So wird demnach eine flächendeckende Analyse der Schadstoffwerte zwar

angestrebt, jedoch nicht erreicht. [UWB19]

Abbildung 1.1: Veranschaulichung einer Messstation, vgl. [UWB19]

1.2 Problemstellung und Zielsetzung

Wie in Abbildung 1.1 zu sehen ist nehmen diese Messstationen sehr viel Platz ein, weshalb es physisch überhaupt nicht möglich ist für ein größeres Gebiet exakte Messungen durchzuführen.

An diesem Punkt setzt diese Bachelorarbeit an. Ziel ist die Entwicklung eines Konzepts zur flächendeckenden Luftqualitätsmessung. Den Platz der Messstationen nehmen dabei Einplatinencomputer, sogenannte *Raspberry Pis* ein, die mit geeigneten Sensoren die Luftqualitätsmessungen durchführen. Der Vorteil dieser Raspberry Pis gegenüber den Messstationen ist ihre Mobilität, welche sie ihrer kompakten Größe zu verdanken haben. Einzelne Messwerte dieser Geräte sind für einen Ort jedoch nicht aussagekräftig genug, weshalb die Daten kollaborativ mittels *Crowdsensing* erhoben werden. Dieser Ansatz ermöglicht es, mit der Hilfe von freiwilligen Crowdworkern an verschiedenen Orten kontinuierlich und parallel aktuelle Live-Daten zu sammeln, ohne dafür Personal einstellen zu müssen. So können ebenfalls Messungen an Orten durchgeführt werden, an denen keine fest installierten Messstationen vorhanden sind und es kann ein

flächendeckendes Monitoring der Schadstoffwerte gewährleistet werden. Hierbei entstehen durch die verschiedenen Crowdworker jedoch Unmengen von Datenströmen, dessen Verarbeitung zuerst auf den Raspberry Pis und anschließend an zentraler Stelle unter verschiedenen Gesichtspunkten geschehen muss, um eine sinnvolle Aussage über die Luftqualität treffen zu können. Für die echtzeitnahe Verarbeitung dieser riesigen Menge an Messwerten eignet sich das *Complex Event Processing* (CEP), mit dem die kontinuierlichen Datenströme analysiert und koordiniert werden können. CEP beschreibt nämlich eine Technologie, mit der anhand von Regeldefinitionen gesuchte Muster in großen Mengen eintreffender Daten detektiert werden können. Die Open-Source CEP Engine *Siddhi* wird hier dafür Anwendung finden.

Das Konzept umfasst dabei die Vorstellung eines sogenannten CEP-Netzwerks. Dieses wird sich aus verschiedenen Komponenten zusammensetzen, die für die Verarbeitung der erfassten Daten bis hin zu ihrer Auswertung hinsichtlich der Luftqualität zuständig sind. Die gewonnenen Informationen können dann genutzt werden, um eine Übersichtskarte der aktuellen Luftsituation zu erstellen. Ein Teil des entwickelten Konzepts wird letztlich auch prototypisch Umgesetzt, um die Durchführungsmöglichkeit der Idee zu demonstrieren.

1.3 Aufbau der Arbeit

Die Arbeit teilt sich in sieben Kapitel auf. Kapitel 2 befasst sich zum einen mit den Grundlagen des Crowdsensing. Dazu schauen wir uns zunächst an, wie der Begriff definiert wird. Außerdem wird aufgezeigt, welche Möglichkeiten es eröffnet und wo es eingesetzt werden kann. Zum Ende dieses Kapitels wird noch auf die in dieser Arbeit verwendete Hardware, das Raspberry Pi eingegangen und es werden Gründe für die Nutzung genannt.

In Kapitel 3 wird die Technologie des Complex Event Processing vorgestellt. Dafür werden die wesentlichen Eigenschaften und Merkmale wie Ereignisse und Regeln des Complex Event Processing vorgestellt und es wird gezeigt, wie die Verarbeitung innerhalb dieser Technologie funktioniert und Informationen aus Datenströmen gewonnen werden können. Der dabei verfolgte Architekturstil der Event-Driven Architektur wird ebenfalls im Zuge dieses Kapitels präsentiert.

Kapitel 4 stellt das Hauptkapitel dieser Arbeit dar und befasst sich mit der Entwicklung des angestrebten Konzepts. Dazu werden zunächst verschiedene Anforderungen bezüglich der Datenerhebung durch die Crowdworker und der Luftqualitätskriterien herausgearbeitet. Im weiteren Verlauf des Kapitels detektieren wir nötige Ereignistypen und erarbeiten ein CEP-Netzwerk, das für die Verarbeitung und Informationsgewinnung zuständig ist.

Die Implementierung der Anwendung wird in Kapitel 6 thematisiert. Zuerst gehen wir näher auf die verwendte CEP-Engine Siddhi ein und schauen uns die Funktionsweise der Engine an. Anschließend wird auf die verwendeten Sensoren eingegangen und erklärt, wie die Daten aus ihnen ausgelesen werden. Daran knüpft die Implementierung der Crowdworker-Anwendung an, von welcher wir uns beispielhaft die verwendeten Regeln zur Mustererkennung durch Siddhi anschauen. Die Server-Anwendung wird auf gleiche weise vorgestellt.

In Kapitel 6 wird die Funktionsfähigkeit des Prototyps und der Anwednung aufgezeigt. Außerdem schauen wir uns an, was für Aussagen wir mit den gewonnenen Erkentnissen treffen können und grenzen die Unterschiede zu konventionellen Messstationen ab.

Am Ende wird in Kapitel 7 auf den Verlauf der Arbeit zurückgeblickt. Mit einer kurzen Bewertung des Ergebnisses und offenen Fragen und Problemen wird die Arbeit dann zum Abschluss gebracht.

2 Hintergrund des Crowdsensing und Raspberry Pi als verwendete Hardware

In diesem Kapitel wird der Begriff *Crowdsensing* als in dieser Arbeit verwendetes Mittel zur Datenerhebung näher beschrieben. Außerdem sollen die Gründe der dafür Verwendete Hardware - ein *Raspberry Pi*, dargelegt werden.

2.1 Crowdsensing

2.1.1 Definition

Der in den letzten Jahren immer mehr in Vordergrund rückende Begriff des *Internet of Things* (IoT) beschreibt die zunehmende Vernetzung von elektronischen Geräten und die damit einhergehende Kommunikation zwischen diesen. Das *Mobile Crowdsensing* (MCS), oder einfach nur Crowdsensing genannt, beschreibt ein neues Wahrnehmungsparadigma, dass sich diesen zunehmenden Trend unter anderem zunutze macht. Die genutzten Geräte des IoT sind hierbei jedoch keine Kaffeemaschinen mit Netzwerkanbindung, sondern beispielsweise Smartphones oder Fahrzeuge. Diese Geräte verfügen oft über verschiedene Sensoren, mit denen es möglich ist das vorhandene Wissen über die reale Welt zu erweitern und dadurch verschiedene Erkenntnisse mit gesellschaftlicher Relevanz zu erschließen. Im Gegensatz zu der Kaffeemaschine verfügen diese Geräte außerdem noch über eine erhöhte Rechenleistung, größere Speicherressourcen, sowie einer starken Kommunikationsfähigkeit. [GYL11] [MZY14] Diese Geräteeigenschaften eröffnen neue Möglichkeiten der Datenerhebung und stellen den Teil des *-sensing*s im MCS dar.

Um jedoch relevante Informationen aus diesen Daten ziehen zu können, ist es nötig die erfassten Daten dieser Geräte räumlich sowie zeitlich zu aggregieren. Dadurch wird ein aussagekräftiges, kollektives Wissen erzeugt, welches auf den einzelnen Beiträgen aus einer großen Menge von Personen, der *Crowd-* basiert. [MZY14]

Es existieren dabei zwei verschiedene Ausprägungen dieses Paradigmas: das *participatory* sowie das *opportunistic sensing*. Ersteres erfordert eine aktive Beteiligung der

Personen, die die zur Datenerhebung genutzten Geräte mit sich führen. Dies kann bswp. in Form von Fotos oder einer einfachen Textnachricht über ein beobachtetes Phänomen geschehen. Das *opportunistic sensing* hingegen erfordert nur eine minimale Beteiligung, die sich meistens auf die Sicherstellung einer Netzwerkverbindung beschränkt und das Gerät im Hintergrund autonom Daten sammelt lässt. [GYL11] Ein Backend-System empfängt und verarbeitet diese Daten dann hinsichtlich des zu untersuchenden Kontexts, woraus beispielsweise eine Kartierung dessen resulitieren kann. Abbildung 2.1 illustriert dieses hinter dem Crowdsensing stehende Konzept.

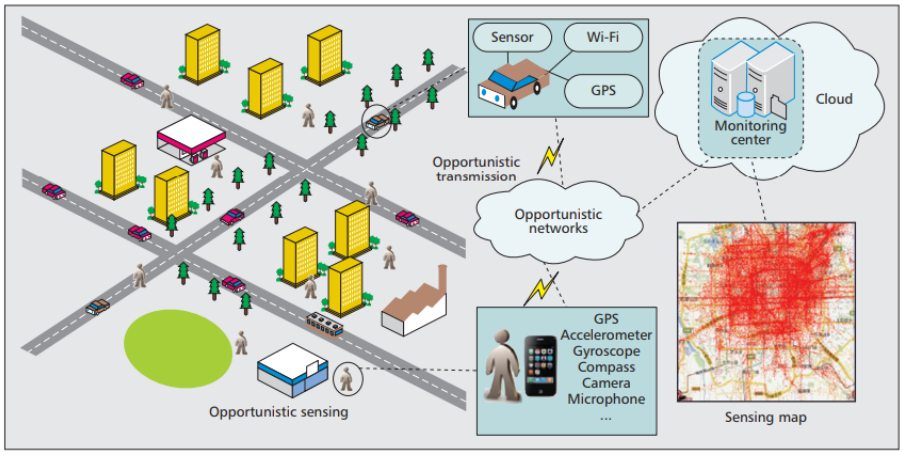

Abbildung 2.1: Veranschaulichung des Crowdsensing-Prinzips, vgl. [MZY14]

2.1.2 Herausforderungen und Chancen

Konventionelle Sensornetzwerke bestehen aus unzähligen Sensor- und Relaisknoten[1], dessen Implementierung und Wartung einen großen Aufwand und hohe Kosten mit sich bringen. Das *CitySee-Projekt* in der chinesischen Stadt Wuxi realisiert ein solches Sensornetzwerk zur CO_2-Überwachung. Insgesamt 100 Sensor- und 1096 Relaisknoten decken dabei eine Fläche von knapp 1km ab. [MMH+12] Als Vergleich: Um für das gleiche Vorhaben eine Fläche von 900km^2 abdecken zu können, was der Größe von Berlin entspricht, würden ca. 90.000 Sensor- und 1.000.000 Relaisknoten benötigt werden. Die Kosten und Pflege für so ein Projekt sind schlicht untragbar.

Crowdsensing jedoch kann dieses Problem umgehen. Aufgrund der Tatsache, dass ein Großteil der benötigten Erfassungsgeräte bereits in unserer Hosentasche oder auf den Straßen im Einsatz sind, ist das benötigte Sensornetzwerk quasi schon vorhanden und die Kosten für die Einrichtung können auf ein Minimum reduziert werden. Außerdem kann so ein Crowdsensing-Netzwerk für verschiedene Problemstellungen verwendet werden,

[1] Dienen zur drahtlosen Datenübertragen zwischen Sensorknoten in einem Sensornetzwerk

wo hingehen eine spezialisierte Sensorinfrastruktur nur für einen bestimmten Sachverhalt konzipiert wird. [GYL11] Straßenkameras und Schleifendetektoren[2] zur Straßen- und Verkehrsüberwachung könnten so gänzliche durch das Crowdsensing abgelöst werden.

Das Problem dabei ist, dass die Einheit der vorhandenen Geräte hinsichtlich ihrer Verfügbarkeit und Fähigkeiten von hochst dynamischer Natur ist und die Qualität der Genauigkeit, Latenz und des Vertrauens in Bezug auf die gesammelten Daten dadurch stark variiert. Dies hängt damit zusammen, dass die Erfassungsgeräte einer Crowd unter anderem über unterschiedliche Energieniveaus und Kommunikationskanäle verfügen. [GYL11] Die Wahrscheinlichkeit, auf einer Straße mehr als zwei Fahrzeuge der gleichen Marke und des gleichen Modells mit der gleichen (zur Analyse ausreichenden) Austtatung zu finden, ist bereits sehr gering und zeigt auf, dass die Datenerfassung über die Verkehrsituation z.B. durch verschiedene Fahrzeuge hinsichtlich der genannten Aspekte auch qualitativ unterschiedliche Daten hervorbringen kann.

Zusätzlich hängt die Verfügbarkeit der Daten mit den Prioritäten des Menschen, der im Besitz des Gerätes ist zusammen. Ein gewisser Anreiz ist deshalb nötig, um die Teilnehmer der Crowd zur Datenerhebung zu animieren, da für sie damit eine Art Aufwand und ggf. Kosten einhergehen. [GYL11] Es müssen Mechanismen dafür entwickelt werden, welche wir hier nicht näher betrachten.
Doch sollten die Nutzer sich zur Datenerhebung bereiterklären, müssen Aspekte wie Datenschutz und Integrität beachtet werden. Jeder Nutzer hat eine andere Vorstellung über das Thema Privatsphäre. Einige haben kein Problem damit, ihre Standortinformationen zu teilen, bei anderen hingegen besteht unter anderem die Sorge, anhand dieser Daten ihren Wohnort oder Arbeitswege zurückverfolgen zu können. Mit Hilfe kryptografischer Verfahren oder durch Datenmanipulation, um nur ein paar Möglichkeiten zu nennen, kann die Privatsphäre der Nutzer geschützt werden. [GYL11] [MZY14]

Da die Teilnehmer in gewisser Weise anonym sind, besteht hier ebenfalls das bereits erwähnte Problem der Integrität. Personen könnten versuchen, Daten zu manipulieren, um das ermittelte Ergebnis zu verfälschen. Daher ist es nötig, solche minderwertigen Daten zu detektieren und ggf. herauszufilter oder zu korrigieren, um weiterhin die gewünschte Ergebnisqualität zu erhalten. [MZY14]

Eine nähere Betrachtung der Problemlösung dieser Schwierigkeiten würde den Rahmen dieser Arbeit sprengen, weshalb sie an dieser Stelle lediglich erwähnt werden und auf die Lösungsnotwendigkeit hingewiesen wird.

Da das Crowdsensing ein Zusammenspiel der menschlichen Mobilität und Aktivität mit elektronischen Erfassungsgeräten beschreibt, existiert hier ein weiteres Potential dieses

[2] In die Fahrbahn eingelassene Kabelschleifen zur Erkennung von Fahrzeugen und Steuerung von beispielsweise Ampeln

Paradigmas. Durch die Nutzung der menschlichen Intelligenz in Kombination mit den Geräten können hochwertige Daten eines semantisch komplexen Sachverhalts erschlossen werden, für die sonst aufwändige Hardware nötig wäre. Es exitieren beispielsweise Konzepte, bei denen freie Parkplätze mittels Ultraschallsensoren, die in den Fahrzeugen integriert sind, erkannt werden sollen. Wenn die Fahrer nun jedoch ihre Smartphones benutzen, um Auskunft über die freien Parkplätze zu geben, können anspruchsvolle Ultraschallgeräte gespart werden. [GYL11]

2.1.3 Anwendungsgebiete

Die Verwendung von MCS bietet sich vor allem in den Bereichen *Infrastruktur*, *Umwelt* und *Soziales* an.

- **Infrastruktur**:
  Neben dem Parkplatzszenario existrieren vor allem Ansätze zu Analyse des Straßen- und Verkehrszustands. So können durch Bewegungssensoren in den Smartphones der Fahrer Rückschlüsse über Straßenschäden geschlossen werden. Nericell ist ein von Microsoft ins Leben gerufene Projekt, dass diesen Ansatz verfolgt um Verkehrsdaten zu sammeln. [MPR08] Das *CarTEL*-Projekt des Massachusetts Institute of Technology (MIT) hingegen verwendet Fahrzeuge als Sensoren zur Stauerkennung. Dabei werden Informationen über Standort und Geschwindigkeit der Fahrzeuge über öffentliche Hotspots an einen zentralen Server gesendet und ausgewertet. [Sho10]
- **Soziales**:
  Aufgrund des aktuell sehr starken Fitness-Trends existieren unzählige Apps, die dazu dienen seinen persönlichen Fortschritt in diesem Bereich zu dokumentieren und mit anderen zu teilen. Dabei werden ebenfalls riesige Datenmengen generiert, die an anderer Stelle genutzt werden können um Erkenntnisse zu gewinnen. Im Bezug auf die Ernährung existieren Methoden zur Analyse von Essgewohnheiten. Dabei werden die Daten von beispielsweise Diabetikern ausgewertet, die ihre tägliche Nahrungszufuhr in solchen Apps eintragen und eine Art Tagebuch führen. Langfristig können daraus dann Erkenntnisse gewonnen werden, die zur Krankheitsbehandlung von Erkrankten dienen können. Sie können auch zur Krankheitsprävention bei gesunden Nutzern verwendet werden, sollten in ihren Daten gewissen Muster auftreten. [GYL11]
- **Umwelt**:
  Sehr viele Möglichkeiten der Nutzung des MCS finden sich im Bereich der Umwelt. So lässt sich der Lärmpegel in einem Gebiet, der durch Fahrzeuge, Menschenmassen oder Ähnlichem verursacht wird mithilfe der Mikrofone an einem Smartphone messen. Ebenfalls ist es möglich, das MCS zur Überwachung des Wasserstands oder der Wasserverschmutzung von Bächen und Flüssen zu verwenden. Durch Fotos oder Textmitteilungen können Personen ihre Beobachtungen teilen,

welche die zuständigen Einrichtungen dann auswerten und nutzen können, um beispielsweise den Wasserstand nach einem starken Regen für einen Flussverlauf abzuschätzen und ggf. geeignet zu reagieren. Selbstverständlich bietet sich die Luftqualitätsüberwachung ebenfalls im MCS äußerst an, weshalb auch diese Arbeit sich dem Thema widmet. Es existieren Möglichkeiten, verschiedene Luftmessewerte mit Hilfe von dafür geeigneten Sensoren zu ermitteln. Mittels Bluetooth können die gemessenen Daten an das Smartphone des Nutzers gesendet werden, welche diese dann an einen zentralen Server zur Auswertung schicken. [GYL11] Der in dieser Arbeit verfolgte Ansatz geht einen anderen Weg und wird im weiteren Verlauf näher beschrieben.

2.2 Raspberry Pi

Entgegen dem in diesem Abschnitt beschriebenen Vorgehen des Crowdsensings, die bereits im Einsatz befindlichen Geräte zur Datenerhebung zu nutzen, werden wir in dieser Arbeit eigene, speziellere Geräte benutzen - **Raspberry Pis**.

Ein Raspberry Pi ist ein Einplatinencomputer und wurde von der britischen *Raspberry Pi Foundation* entwickelt. Ursprünglich waren diese Geräte als äußerst günstige Alternative zu den üblicherweise teuren Heimcomputern gedacht, um das Interesse an der Entwicklung in der Informatik zu fördern. Heute sind die Raspberry Pis so stark, dass auf ihnen die verschiedensten Anwendung ausgeführt werden können. [Fou20b] In dieser Arbeit wird ein **Raspberry Pi 4 Modell B** verwendet. Dieses Gerät verfügt über **4 GB** SDRAM und einem **64-bit Quad Core** Prozessor mit einer Leistung von **1,5GHz**. Als Festplattenersatz kommt eine Micro-SD Karte mit **64 GB** zum Einsatz. Zusätzlich sind noch W-Lan und Bluetooth Empfänger in dem Gerät verbaut. [Fou20c] Wie auf handelsüblichen Heimcomputern kann auch auf einem Raspberry Pi ebenfalls Java, bzw. eine JRE installiert werden. Dadurch können Java-Programme auch für dieses Gerät programmiert und auf diesem ausgeführt werden. Bei der Entwicklung auf dem Gerät selbst zeigen sich jedoch bereits seine Grenzen auf, da allein der Compiliervorgang zwar möglich, jedoch sehr Zeitaufwändig ist. Daher sollten die Programme auf einem konventionellen Rechner entwickelt werden und auf dem Raspberry lediglich ausgeführt werden.

Abbildung 2.2: Raspberry Pi, vgl. [Fou20a]

In Abbildung Abbildung 2.2 ist ein solches Gerät zu sehen. Die kompakte Größe in Kombination mit ihrer Leistungsfähigkeit sind bereits gute Gründe für uns, den Raspberry Pi als Gerät zur Datenerhebung zu verwenden. Ein weiterer und ausschlaggebender Grund ist die Möglichkeit, Sensoren an den Raspberry Pi über sogenannte GPIO-Pins (general-purpose input/output) anzuschließen, um gewisse Phänomene zu beobachten. In unserem Fall sind dies Sensoren zur Messung der Luftverschmutzung und GPS-Sensoren. Smartphones beispielsweise können uns zwar Standortinformationen liefern, jedoch können sie nicht mit den hier benötigten anwendungsspezifischen Sensoren zur Luftqualitätsmessung ausgestattet werden. Mit den Raspberry Pis können wir das Problem der großen Messstationen umgehen und eine Vielzahl unserer eigenen Messstationen quasi in der Hand mitführen. Um die gemessenen Werte verarbeiten können, sind die Raspberry Pis ebenfalls in der Lage das Complex Event Processing auszuführen, welches in Kapitel 3 ausführlicher beschrieben wird.

3 Grundlagen des Complex Event Processing und der Event-Driven Architecture

In diesem Kapitel werden die grundlegenden Begriffe des Complex Event Processing beispielhaft vorgestellt. Zur Verwendung dieser Technologie muss ein ereignisgesteuerter Entwurfsstil vorfolgt werden, die sogenannte *Event-Driven Architecture* (EDA). Die Grundlagen dessen werden hier ebenfalls illustriert, um das Verständnis für die voranschreitende Arbeit zu stärken.

3.1 Motivation und Definition

Mit der stetig zunehmenden Digitalisierung wächst auch der im Netz vorhandene Berg an Daten. Social Media, Monitoringsysteme, Wirtschaftsmärkte oder das bereits in Kapitel 2.1 erwähnte Internet of Things sind nur einige der Quellen, die kontinuierliche Datenströme erzeugen. Durch die Analyse und Synthese dieser Daten können Rückschlüsse und Zusammenhänge über verschiedene Sachverhalte erschlossen werden. So können beispielsweise Unternehmen Risiken früh erkennen und Chancen rechtzeitig nutzen, um Geschäftsprozesse zu optimieren und ihre Position am Markt zu stärken. [Hed17] [BD15] Damit solche Reaktionen jedoch durchgeführt werden können, muss der Anspruch bei einer zeitnahen Extraktion der für einen selbst relevanten Informationen liegen.
Eine Möglichkeit der echtzeitnahen Datenstromverarbeitung bietet das Complex Event Processing. Es beschreibt eine Technologie, mit der Ereignisse als Repräsentation der einströmenden Daten mittels Mustererkennung (*Event Pattern Matching*) „identifiziert, transformiert, korreliert und bewertet“ [BD10] werden können. Mit Hilfe von Regeln, die durch spezielle Ereignisanfragesprachen definiert werden, wird das riesige Volumen von Ereignissen nach diesen Mustern durchsucht und der Systemfluss (ereignis)-gesteuert.

3.2 Ereignisse

Im Zusammenhang des CEP beschreibt ein Ereignis ein jedes Geschehnis, das zu einer Zustandsänderung in der zu untersuchenden Anwendungsdomäne führt. Dabei können

diese Ereignisse in mehrere Abstraktionsstufen mit unterschiedlichem Informationsgehalt klassifiziert werden. Technische Ereignisse wie einzelne Messungen von Sensoren sind meistens nicht so aussagekräftige wie die Veränderung eines einzelnen Warenbestands auf einen bestimmten Grenzwert, was ein Geschäftsereignis darstellt. [BD10] Der Nutzen eines Ereignisses hinsichtlich seines Informationsgehalts hängt somit von den Daten ab, die es trägt. In der Regel enthalten einfache Ereignisse neben den für die Weiterverarbeitung relevanten Informationen (*payload*) noch eine ID zur Unterscheidung von Ereignissen gleichen Typs, einen Zeitstempel des Ereigniseintritts, sowie Daten über den Übersprung des Vorkommnisses. [BD15] Solche Ereignisse werden auch als **atomare Ereignisse** bezeichnet. Sollte der Informationsgehalt, wie bereits bei den Sensordaten erwähnt, nicht aussagekräftig genug sein, ist es nötig diese atomaren Ereignisse mit geeigneten Mitteln des CEP zu sogenannten **komplexen Ereignissen** zu aggregieren (siehe Abbildung 3.1).

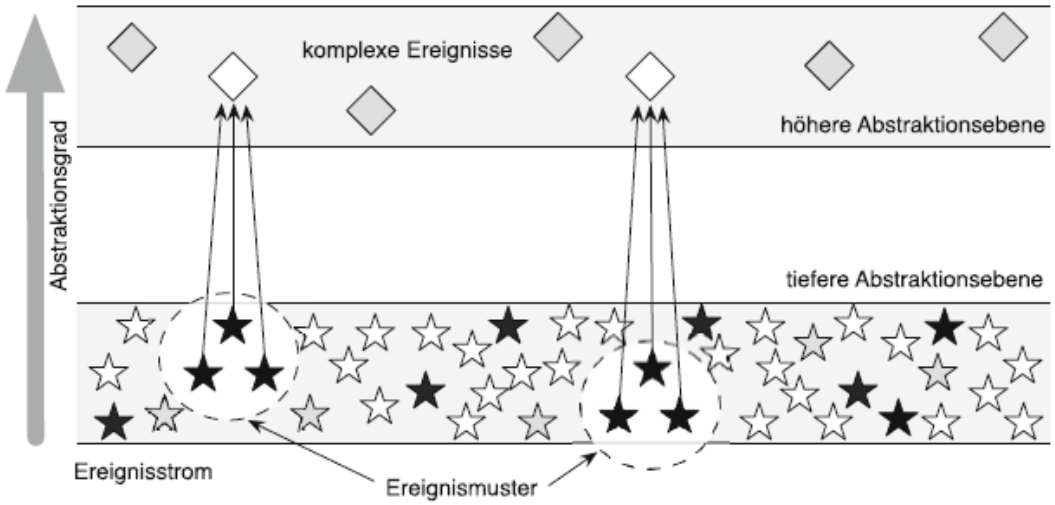

Abbildung 3.1: Erhöhung des Abstraktionsgrad durch Ereignisaggregation, vgl. [BD15]

Zur beispielhaften Erläuterung sehen wir uns die sensorbasierte Energieüberwachung in Gebäuden am Beispiel einer Universität [1] an. In diesem Szenario existieren die atomaren Ereignisse:

- Bewegungsereignis - beschreibt die gemessene Bewegung durch Menschen in einem Raum
- Temperaturereignis - beschreibt die gemessene Temperatur in einem Raum
- Kontaktsensorereignis - beschreibt das Öffnen/Schließen einer Tür/eines Fensters

Die Messung eines niedrigen Temperaturwertes in einem Raum allein lässt noch keinen Rückschluss daraus ziehen, ob es nötig ist die Heizung anzustellen. Erst durch die Kombination mit einem Bewegungs- und Kontaktsensorereignis wird klar, dass sich

[1] Angepasst und entnommen aus [BD13]

Menschen in dem Raum befinden und eine Temperaturerhöhung durch das automatische Anstellen der Heizung sinnvoll ist. Demnach resultiert daraus ein komplexeres *Temperaturerhöhungsereignis* mit einem höheren Abstraktionsgrad.

3.3 Konzepte der Ereignisverarbeitung

Die Generierung solcher komplexen Ereignisse erfolgt durch die Analyse der kontinuierlich eintreffenden atomaren Ereignisse. Dabei werden Muster definiert, die einen fachlichen Sachverhalt in der realen Welt darstellen. Sobald in den Ereignisströmen solche Muster detektiert werden, kann ein höheres Ereignis daraus abstrahiert werden. Dies nennt man *Event Pattern Matching* [BD15].

3.3.1 Ereignisregeln

Durch die Definition von Regeln mit einer speziellen *Event Processing Language* (EPL), können die gesuchten Muster aus dem Anwendungskontext bestimmt werden. Diese Muster beschreiben eine Voraussetzung die erfüllt sein muss, um wie in unserem Beispiel der Temperaturerhöhung, eine bestimmte Reaktion auszuführen. Eine Regel setzt sich somit aus einem **Bedingungs-** und einem **Aktionsteil** zusammen. Der Vorteil eines solchen regelbasierten Ansatzes ist, dass Anwendungen die ein solches Vorgehen nutzen, leicht erweiter- und wartbar sind. Sollten sich Änderungen in der Anwendungsdomäne ergeben, können Regeln einfach hinzugefügt oder angepasst werden, ohne drastische Änderungen an den Strukturen des Systems vornehmen zu müssen.

Da ein gesuchtes Schema aus der Realität meist komplexen Anforderungen i.S.v. zeitlichen und kausalen Zusammenhängen gerecht werden muss, existieren verschiedene Konzepte zur Regeldefinition, die im Folgenden anhand unseres Beispielszenarios kurz erläutert werden. [BD15]

Sprachkonzepte

Zum einen existieren verschiedene Operatoren zur Regeldefinition:

- **boolsche Operatoren** (A∧B / A∨B): Wie aus der Informatik bekannt, haben diese logischen Operatoren hier dieselbe Bedeutung. Um ein Temperaturerhöhungsereignis ET auszulösen, müssen bestimmte Instanzen des Temperaturereignisses T und des Bewegunsereignisses B sowie Kontaktsensorereignisses K im Ereignisstrom auftreten (T∧B∧K) . Dabei spielt es keine Rolle, welches Ereignis zuerst auftritt. Die Verwendung des ∨-Operators erfolgt analog, falls nur eines der Ereignisse benötigt wird. [BD15]

- **Sequenzoperator** (A $\rightarrow$ B): Angenommen, für den nachliegenden Anstoß eines Vorgangs als Reaktion ist es von Bedeutung, in welcher Reihenfolge gewisse Ereignisse im Datenstrom detektiert werden. Dazu wird der Sequenzoperator verwendet. Mit ihm wird festgelegt, dass auf ein Ereignis A ein Ereignis B folgen **muss**, unabhängig davon wie viele Ereignisse anderen Typs dazwischen im Ereignisstrom erfasst werden. [BD15]
- **Negationsoperator** ($\neg$A): Ebenfalls kann es nötig sein, das bestimmte Ereignisse in einem Muster nicht auftreten dürfen. Benutzt wird dies jedoch lediglich in Kombination mit einem zeitlichen oder quantitativen Bezug, auf welchen wir im nächsten Abschnitt eingehen werden. [BD15]

Windows

Einfache Operatoren reichen bei der Regeldefinition nicht aus, um spezielle Schemata zu erkennen. Die Beschränkung auf eine genaue Anzahl der erfassten Ereignisse oder auch ein bestimmter zeitliche Rahmen in dem sie auftreten spielt in den meisten Fällen eine Rolle. *Sliding Windows* sind ein Mittel des CEP, solche Eingrenzungen vorzunehmen. Zusätzlich wird damit durch Ereignisreduzierung ein wirkungsvolles CEP gewährleist, da die Persistierung sowie Verarbeitung des gesamten Datenvolumina ohnehin nicht möglich wäre. [BD15]
Man unterscheidet dabei zwischen zwei Arten von Sliding Windows:

- **Zeitfenster**: Wenn wir in unserem Eingangsbeispiel die Heizung nach Ende einer Veranstaltung in einem Saal automatisch ausschalten wollen, müssen wir einen bestimmten Zeitraum betrachten in dem keine Bewegung wahrgenommen wurde. Eine Regel könnte wie folgt aussehen: (K $\rightarrow$ T) $\rightarrow$ $\neg$B[win:time:10min]
  Diese Regel besagt, dass auf ein Kontaktsensorereignis einer geschlossenen Tür für einen bestimmen Saal ein Temperaturereignis einer erhöhten Temperatur dieses Saals folgen muss. Daraufhin darf dort in einem Zeitraum von 10 Minuten kein Bewegungsereignis eintreffen, um sicher zu stellen dass sich niemand mehr im Raum befindet und die Heizung abgeschaltet werden kann. [BD10]
- **Längenfenster**: In manchen Fällen ist es nötig eine bestimmte Anzahl an Vorkommnissen hinsichtlich gewisser Aspekte zu betrachten, wofür sich Längenfenster eignen. Sobald eine festgelegte Anzahl an Ereignissen eingetroffen ist, wird das Ereignis das als erstes eingetroffen ist aus dem Betrachtungsraum entfernt, sollte ein neues eintreffen. [BD10]

Aggregationen

Um die in einem Intervall betrachteten Ereignisse gleichen Typs in Beziehung setzen zu können, werden Aggregationsfunktionen benutzt. Mit ihnen ist es möglich Dinge wie Durchschnitt (**avg()**), Summe (**sum()**), sowie Maximum (**max()**) und Minimum

(**min()**) zu ermitteln. Die Berechnung kann unter verschiedenen Aspekten geschehen, auf welche wir hier nicht weiter eingehen. [BD10]

Beispielhafte Ereignisanfragesprache

Im folgenden schauen wir uns eine einfach Regel für ein Muster in unserem Beispielszenario an. Die verwendete Pseuedo-Ereignisanfragesprache ist an die in dem Buch [BD10] verwendete Ereignisanfragesprache angelehnt und für die in dieser Arbeit definierten Regeln angepasst.

Wir wollen in unserem Beispielszenario für jeden Vorlesungssaal die Durchschnittlichen Raumperaturen der letzten 4 Stunden ermitteln. Wir definieren dafür folgende Regel mit unserer Pseudo-Ereignisanfragesprache:

```
CONDITION: (RoomTempEvent as r)[win:time:batch:4h]
GROUP BY: r.roomNo

ACTION: create AvgRoomTempEvent(roomNo = r.roomNo, avgTmp =
avg(r.temp))
```

RoomTempEvents stellen hier Ereignisse dar, die für Temperaturmessungen in verschiedenen Räumen stehen. Wir erwarten also solche Ereignisse in einem Zeitraum von vier Stunden. Dafür definieren wir uns ein Window (Abschnitt 3.3.1). Dieses hat die Besonderheit, dass es als Batch-Window markiert ist, wodurch die Berechnung erst nach Ablauf der 4 Stunden ausgeführt wird. Um die Durchschnittstemperaturen pro Saal zu erhalten, gruppieren wir die eintreffenden Ereignisse anhand ihrer Saalnummern. Am Ende erzeugen wir uns mit den gewonnenen Informationen ein neues Ereignis vom Typ AvgRoomTempEvent, das Daten über die Durchschnittstemperaturen aller Säle der letzten vier Stunden trägt.

3.3.2 Event Processing Agents

Aufbau

Das CEP in einer EDA repräsentiert sich durch *Event Processing Agents* (EPA). In ihnen werden die eben vorgestellten Regeln zur Mustererkennung festgelegt. Abbildung 3.2 zeigt die wesentlichen Komponenten eines solchen EPA.

Neben den **Ereignisregeln** ist ebenfalls ein **Ereignismodell** notwendig, mit dem die Regeln referenziert werden. Dort wird das nötige Hintergrundwissen zu den gesuchten Ereignistypen festgelegt, wodurch gleichzeitig die Verantwortlichkeiten eines EPA

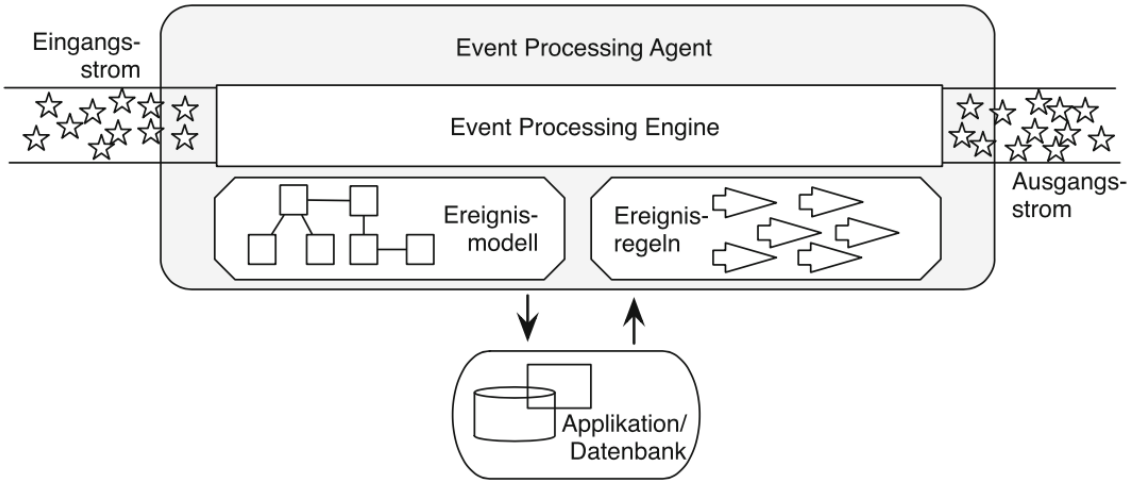

Abbildung 3.2: Zusammensetzung eines EPA, vgl. [BD15]

bestimmt werden. Sollte das Wissen über die Ereignistypen nicht ausreichend im Ereignismodell definiert sein, ist es ebenfalls möglich die eingehenden Ereignisse durch Zugriff auf andere Anwendungen wie eine Datenbank mit weiterem Kontextwissen anzureichern. [BD15]

Um die in einem EPA hinterlegte Regelbasis auch nutzen zu können, wird eine **Event Processing Engine** benötigt. Sie ist die treibende Kraft eines EPA und führt die Mustererkennung auf den einströmenden Datenfluss aus. [BD15] Die in dieser Arbeit verwendete CEP-Engine „Siddhi“ wird in Kapitel 6.1 näher vorgestellt.

Aufgaben

Die grundlegenden Aufgaben von EPAs unterscheiden sich nicht voneinander und folgen einer einhetlichen Aufgabenstruktur, welche in Abbildung 3.3 dargestellt ist.

In einem **Monitoringschritt** wird der Datenstrom nach Ereignissen abgesucht, die der EPA für seine Bearbeitung benötigt. Andere Ereignisse werden hierbei ignoriert und die Ereignismenge so reduziert. Außerdem kann es vorkommen, dass Daten fehlerhaft und somit nicht für die Weiterverarbeitung relevant sind. In unserem Szenario der Energieüberwachung kann es beispielsweise passieren, dass ein Sensor eine unrealistische Raumtemperatur von $-50\,°C$ übermittelt, was auf eine mögliche Fehlfunktion des Sensor schließen lassen könnte. Die genauen Spezifikationen solcher Einschränkungen entnimmt der EPA aus den *Constraints* des Ereignismodells. [BD10] [BD15]

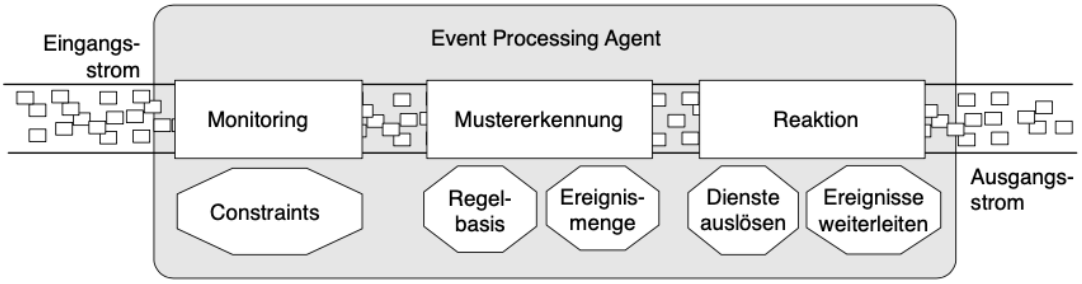

Abbildung 3.3: Aufgaben und Struktur eines EPA, vgl. [BD10]

Die nachfolgenden Schritte bilden den aus den Ereignisregeln bereits bekannten Bedingungs- und Aktionsteil als Vorgang im EPA ab. Sobald ein Muster mit der gegebenen Regelbasis in der zu untersuchenden Ereignismenge detektiert wird, erfolgt eine Reaktion in einem nachgelagerten Anwendungssystem, sofern die Ereignisse nicht noch an einen weiteren EPA weitergeleitet werden müssen. [BD10]

3.4 Event-Driven Architecture

Im Gegensatz zu dem klassischen Ansatz des sequentiellen, ablauforientierten Vorgehens in Anwendungssystemen, wird der Ablauf in einem ereignisgesteurten System, wie der Name schon sagt, über die Kommunikation mit Ereignissen gelenkt. Ereignisse bilden somit das Fundament in einer EDA, weshalb auch das CEP eine zentrale Rolle bei diesem Architekturstil spielt. [BD15] In der Regel durchläuft eine EDA kontinuierlich die drei Schritte *erkennen*, *verarbeiten* und *reagieren*, welche sich auch in den logischen Strukturen einer solchen ereignisgestuerten Anwedung wiederspiegeln (siehe Abbildung 3.4).

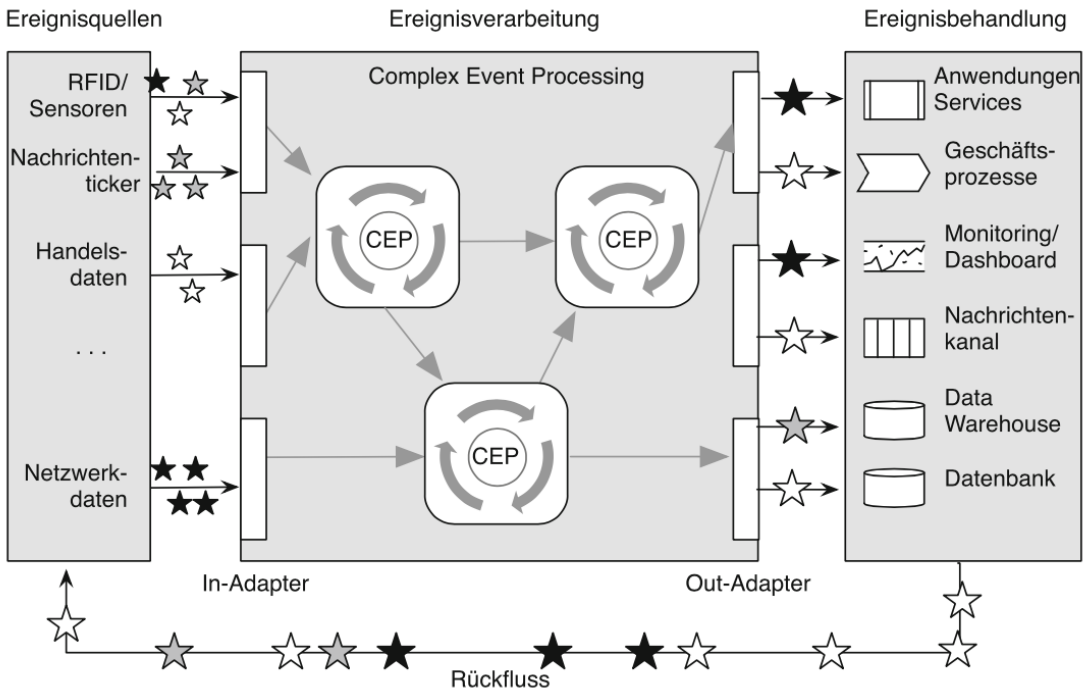

Abbildung 3.4: Schichten einer Event-Driven Architecture, vgl. [BD10]

Schichten

Um Ereignisse *erkennen* zu können, werden **Ereignisquellen** benötigt. Diese können z.B. Sensoren wie in unserem Beispielszenario die Temperatur-, Bewegungs- und Kontaksensoren sein. Jedoch kann auch der Zugriff auf einen Raumbelegungsplan durch eine Serviceanwendung als Quelle fungieren. So kann beispielsweise unsere in Abschnitt 3.3.1 definierte Regel mit einem Raumbelegungsereignis ergänzt werden. Es existieren noch unzählige weitere Quellarten, aber unabhängig davon um was für Ereignisquellen es sich konkret handelt, sie alle haben die Aufgabe aus den erfassten Daten geeignete Ereignisobjekte zu erzeugen und sie in den nachgelagerten Schritt der **Ereignisverarbeitung**

zu übergeben. Oft ist dazu noch ein **In-Adapter** nötig der die Daten in ein einheitliches Format zur Verarbeitung überführt, da die Ereignisquellen unterschiedliche Formate unterstützen. [BD10]

Die **Ereignisverarbeitung** stellt das Herzstück einer EDA dar. In ihr ist das bereits betrachtete CEP mit seinen Konzepten der Verarbeitung angesiedelt. Da ein ereignisgesteuertes System in der Regel eine große Zahl unterschiedlicher Muster in den Ereignisströmen sucht und die Ereignisse nach Detektion mehrere Verarbeitungschritte durchlaufen müssen, existieren in dieser Schicht sogenannte *Event Processing Networks* (EPN) bestehend aus mehrere EPAs. Die einzelnen Komponenten eines Netzwerks kapseln dabei jeweils einen kleinen Teil der verwendeten Regelmenge und führen unterschiedliche Verarbeitungsschritte durch. Damit stellen sie in ihrem Netzwerk für nachgelagerte EPAs gleichzeitig auch eine Ereignisquelle dar. Die Kommunikation erfolgt hier über ein asynchrones *Publish/Subscriber-Prinzip*, bei dem ein oder mehrere Agenten quasi das Thema des für sie relevanten Ereignistyps abonnieren und diese dann weiterverarbeiten können, sollten neue Ereignisse dieses Typs durch einen anderen Publisher-EPA veröffentlicht werden. Durch die Verwendung eines solchen Netzwerks steigt die Effizienz der Musterbearbeitung sowie die Wartbarkeit der gesamten Anwendung enorm, da jede Komponente einen kleineren Teil der zu verarbeiteten Gesamtlast übernimmt und bei möglichen Veränderungen in der Anwendungsdomäne angepasst werden kann. [BD10]

Die Ansteuerung der Heizung in unserem Beispiel würde in der **Ereignisbehandlung** angesiedelt sein. Alle Reaktionen die als Folge der abschließenden Ereignisverarbeitung ausgeführt werden sollen, werden in dieser Schicht angestoßen. Diese Reaktionsdienste können das interne Ereignisformat jedoch nicht verwenden, weshalb ein **Out-Adapter** nötig ist um die Ereignisse in das von den Diensten benötigte Format zu überführen. An dieser Stelle kann auch ein wie in Abbildung 3.4 dargestellter *Rückflüss* der Ereignisse in die erneute Ereignisverarbeitung stattfinden. [BD10]

4 Konzeption einer crowdbasierten Luftqualitätsmessung mit CEP

4.1 Einführung und Vorstelung des Szenarios

Wie wir bereits festgestellt haben, führen herkömmliche Messstationen lediglich an durch Richtlinien festgelegten Orten Messungen zu Schadstoffwerten in der Luft aus. Anhand dieser Messungen werden dann durch Schätzungen, Aussagen über die Luftqualiät in einem weitläufigen Gebiet gemacht. Doch da es sich dabei immer noch nur um Schätzung handelt, sind diese Aussagen für Orte in dem betreffenden Gebiet, an denen keine Messstationen vorhanden sind nicht genau. Für ein genaues Monitoring in einem Gebiet wären riesige Sensornetzwerke nötig, mit denen sehr hohe Kosten verbunden sind. Im folgenden entwickeln wir deshalb ein Konzept, das unter Verwendung des Crowdsensings in Kombination mit CEP diese Probleme umgeht. Dafür betrachten wir folgendes Szenario:

Die Luftqualität in einem Gebiet soll hinsichtlich verschiedener Punkte flächendeckend analysiert werden. Um das Crowdsensing zu ermöglichen, stehen uns dafür sogenannte *Crowdworker* zur Seite, die sich mit moderater Geschwindigkeit willkürlich durch ein zu untersuchendes Gebiet bewegen. Ausgestattet sind diese Crowdworker mit den in Abschnitt 2.2 beschriebenen Raspberry Pis, an die verschiedene Sensoren zur Messung der Schadstoffe in der Luft angeschlossen sind. Zusätzlich verfügen die Geräte über einen GPS-Sensor, um die Position der Crowdworker in dem Gebiet zu erfassen. Zur Verarbeitung der ermittelten Daten ist auf dem Raspberry Pi eine Anwendung implementiert, die das CEP verwendet. Damit ist es möglich, bereits auf dem Raspberry Pi Sensordaten vorzufiltern und zu verwerten. Diese vorverarbeiteten Daten werden dann an einen Server zur zentralen Auswertung geschickt, um Erkentnisse über die Luftsituation in dem Gebiet zu gewinnen und auf diese geeignet zu reagieren. Zur zentralen Verarbeitung ist jedoch eine Koordination der von den Crowdworkern gemessenen Daten nötig, wofür ebenfalls CEP verwendet wird.

Dieses Szenario ist ebenfalls in Abbildung 4.1 dargestellt und beschreibt den grundlegenden Ablauf, welcher mit dem zu entwickelten Konzept angestrebt wird. Zunächst müssen wir dazu Anforderungen an unsere Crowdworker definieren, um verwertbare Sensordaten von ihnen zu erhalten. Im weiteren Verlauf dieses Kapitels werden wir

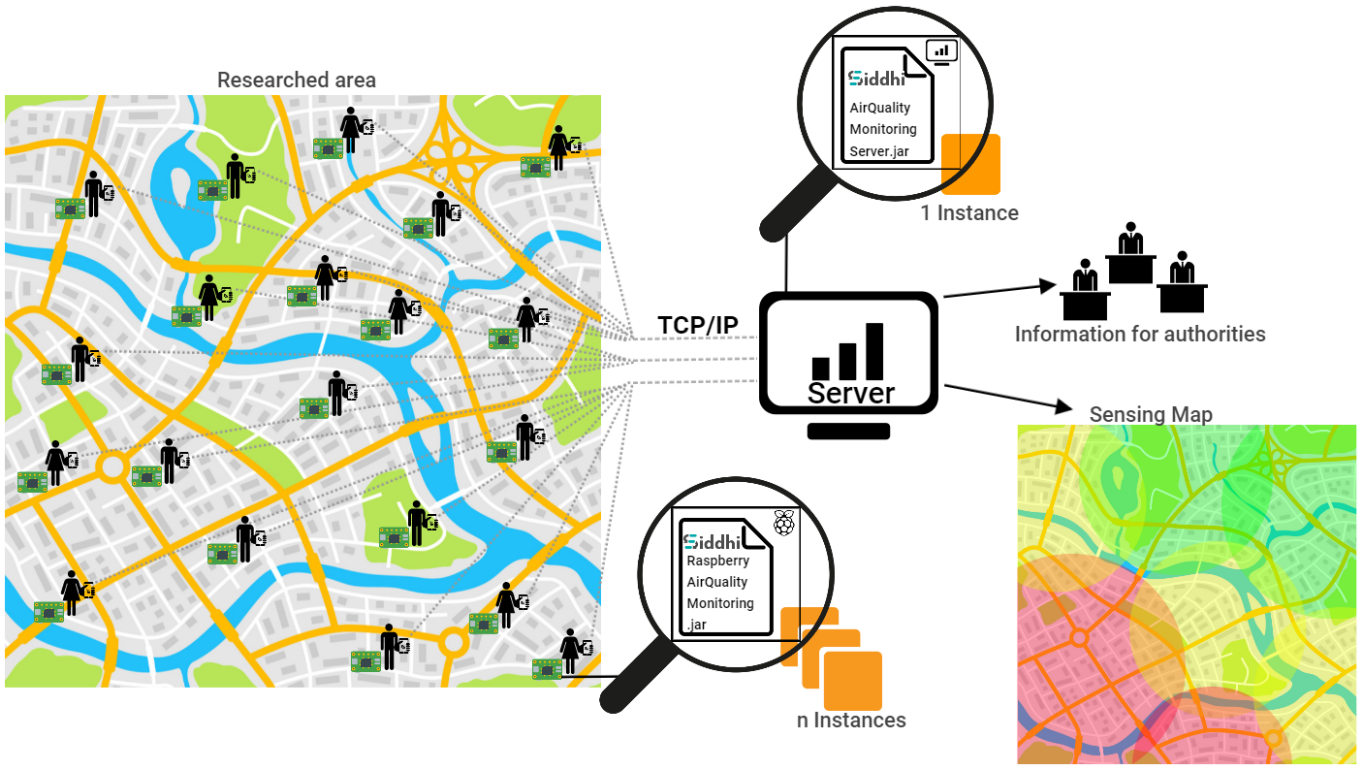

Abbildung 4.1: Physikalisches Szenariomodell

zusätzlich Kriterien der Luftqualität definieren, die es zu detektieren gilt. Diese Kriterien werden uns als Grundlage zur Musterdefinition dienen, welche unsere Anwendung nutzen wird um die gemessenen Datenströme zu verarbeiten. Zur Verarbeitung definieren wir EPAs, die als Teil eines Event-Processing Netzwerks verschiedene Aufgaben übernehmen. Das Netzwerk verteilt sich dabei auf die Einheit der Crowdworker und einer zentralen Server-Komponente. Die verarbeiteten Daten sollen vom Server schließlich genutzt werden, um eine visuelle Darstellung über die Luftsituation zu erstellen und ggf. Auskunft über gesundheitsbedenkliche Schadstoffwerte in einem Gebiet zu geben. Diese können an Einrichtungen weitergeleitet werden, um möglicherweise notwendige Maßnahmen zu treffen.

4.2 Anforderungen an die Crowdworker

Bevor wir die von unseren im Einsatz befindlichen Crowdworker ermittelten Sensordaten verarbeiten zu können, müssen wir unterschiedliche Rahmenbedingungen zur Datenerhebung festlegen. Diese sollen zum einen beschreiben, unter welchen Bedingungen die Crowdworker die Raspberry Pis betreiben können und die gemessenen Daten an den Server senden könne. Andererseits wird auch festgelegt, unter welchen Umständen die

gemessenen Daten für uns keinen Wert haben. Im folgenden werden diese Kriterien aufgeführt:

- **Stromversrorgung des Raspberry Pi**: Da die Crowdworker mobil sind, steht ihnen während den Messungen keine Steckdose zur verfügen, an die das Netzteil des Raspberry Pis angeschlossen werden kann. Eine recht einfache Möglichkeit besteht darin, den Raspberry Pi an eine Power Bank anzuschließen. Diese verfügen in der Regel über einen 5V Ausgang, der für den Betrieb des Raspberry Pis benötigt wird. Es existieren mit Sicherheit weitere Möglichkeiten dafür, doch welche Alternative letztlich gewählt wird, soll keinen Einfluss auf die Messungen nehmen. Hier soll lediglich erwähnt werden, dass eine externe Stromquelle benötigt wird.
- **Datenübermittlung**: Abbildung 4.1 zeigt, dass die gesendeten Daten der Raspberry Pis mittels TCP-Verbindung an die IP-Adresse des Servers adressiert werden. Dieser wartet auf eintreffende Verbindungsanfragen an dem jeweiligen Socket. Für die Datenübertragung benötigen die Geräte jedoch eine Internetverbindung. Sollten öffentliche Hotspots verfügbar sein, können diese genutzt werden. Oft fehlen diese jedoch, weshalb eine andere Möglichkeit benötigt wird. Falls die Crowdworker über ein Smartphone verfügen, können diese einen Hotspot ihrer Mobilfunkverbindung bereitstellen, über den die Raspberry Pis ihre gemessenen Daten dann übermitteln können. Das Problem dabei ist, dass für die Crowdworker ggf. Kosten im Sinne von verbrauchtem Datenvolumen oder zusätzlichen Verbindungskosten auftreten können. Sollten die Crowdworker demnach nicht bereit sein, ihre persönliche Mobilfunkverbindung zur Verfügung zu stellen oder überhaupt kein Smartphone besitzen, besteht weiterhin die Möglichkeit einen Surfstick zu verwenden. Solche Surfsticks verfügen über ein UMTS- oder LTE-Modem, mit dem sie sich wie ein Smartphone mit dem Mobilfunknetz verbinden können.
- **Geschwindigkeit / Fortbewegung**: Um eine korrekte Funktionsweise der an den Raspberry Pis angeschlossenen Sensoren zu gewährleisten, müssen wir unter anderem sicherstellen, die Crowdworker zu Fuß unterwegs sind. Selbstverständlich können wir nicht davon ausgehen, dass sie während der gesamten Zeit, in der das Raspberry Pi Daten sammelt nicht zwischenzeitlich mit dem Auto unterwegs sind oder öffentliche Verkehrmittel nutzen. Deshalb müssen wir zumindest die Daten herauszufiltern, die nicht während der Fortbewegung zu Fuß ermittelt wurden.
- **Außenluft**: Das in dieser Arbeit zu entwickelnde Konzept soll sich mit der Luftqualitätsmessung der Außenluft befassen. Demnach müssen wir Messungen erkennen, die innerhalb von Gebäuden durchgeführt werden, um diese aus unserem Ereignisfluss zu entfernen. Diese müssten nämlich zur Innenraumluftanalyse gezählt werden, welche wir in dieser Arbeit nicht behandeln.

4.3 Anforderungen an die Luftqualität

Zunächst müssen wir uns anschauen, welche Schadstoffwerte wir untersuchen müssen. Aufschluss darüber gibt uns das Umweltbundesamt. Dieses trifft Aussagen über die in unserer Luft vorhandenen Schadstoffe in Form eines *Luftqualitätsindex* (LQI). Der LQI setzt sich aus dem Anteil der drei Schadstoffewerten Feinstaub (PM_{10}), Ozon (O_3) und Stickstoffdioxid (NO_2) zusammen. Abbildung 4.2 zeigt die Zuordnung der Messwerte zum LQI.

Index	Stundenmittel NO_2 in µg/m³	stündlich gleitendes Tagesmittel PM_{10} in µg/m³	Stundenmittel O_3 in µg/m³
sehr schlecht	› 200	› 100	› 240
schlecht	101-200	51-100	181-240
mäßig	41-100	36-50	121-180
gut	21-40	21-35	61-120
sehr gut	0-20	0-20	0-60

Abbildung 4.2: Bewertungsgrundlage des LQI anhand der gemessenen Schadstoffwerte, vgl. [Umw19]

Dabei gibt die am höchste gemessene Konzentration den Wert des LQI an. [Umw19] Nehmen wir zum Beispiel an, der gemessene PM_{10}-Wert liegt bei $50\mu g/m^3$ (mäßig - gelb), während die Werte für NO_2 bei $10\mu g/m^3$ (sehr gut - türkis) und für O_3 bei $70\mu g/m^3$ (gut - grün) liegen. Der LQI würde demnach *mäßig* entsprechen. Sollten nicht über jeden dieser Werte Daten vorhanden sein, also mindestens einer der Schadstoffwerte fehlen, ist dies entsprechend gekennzeichnet.
Zusätzlich zu diesen Schadstoffen existieren zwei weitere, welche wir ebenfalls in unser Konzept einbeziehen wollen. Dazu gehören neben dem PM_{10}-Feinstaub noch der feinere und gefährlichere $PM_{2,5}$-Feinstaub und das Kohlenmonoxid (CO).

Da in dieser Arbeit das Potential des CEP zusammen mit Crowdsensing in der Luftqualitätsanalyse aufgezeigt werden soll, werden wir nicht auf die Emittenten oder gesundheitlichen Risiken dieser Schadstoffe eingehen, da dies den Rahmen der Arbeit sprengen würde und für das Voranschreiten der Arbeit auch nicht notwendig ist. Jedoch müssen wir in unser Konzept ebenfalls mit einbeziehen, dass neben den reinen LQI-Kriterien die Schadstoffe auch über verschiedene, festgelegte Grenzwerte verfügen, die über unterschiedliche Zeiträume beobachtet werden müssen und nur bedingt überschritten werden dürfen. Nachfolgend halten wir diese zusätzlichen Kriterien fest:

- **Feinstaub (PM_{10})**: Der PM_{10}-Wert wird als stündlich gleitender Tagesmittelwert berechnet und darf nicht öfter als 35mal in einem Jahr im Tagesmittel über $50\mu g/m^3$ liegen. Daraus ergibt sich ein erlaubter Jahresmittelwert von $40\mu g/m^3$. [Umw20b]

- **Feinstaub ($\mathbf{PM_{2,5}}$)**: Für diesen Wert muss ausschließlich der Jahresmittelwert betrachtet werden. Dieser liegt bei $25\mu g/m^3$ und darf nicht überschritten werden. [Umw20b]
- **Ozon ($\mathbf{O_3}$)**: Für den Ozonwert wird das Stundenmittel betrachtet. Außerdem gilt hier eine Informationspflicht sowie eine Verhaltensempfehlung an die Bevölkerung, sollten Stundenmittelwerte von $180\mu g/m^3$ und $240\mu g/m^3$ erreicht werden. Im wesentlichen darf hier jedoch ein Mittelwert über acht Stunden an einem Tag nicht öfter als 25mal im Jahr über $120\mu g/m^3$ liegen. [Umw20d]
- **Kohlenmonoxid (CO)**: Ähnlich wie bei Ozon, wird hier ein 8-Stunden-Mittelwert eines Tages betrachtet. Eine Überschreitung von $10\mu g/m^3$ ist nicht erlaubt. [Umw20c]
- **Stickstoffdioxid ($\mathbf{NO_2}$)**: Betrachtet wird hier erneut der Stundenmittelwert. Eine Überschreitung von $200\mu g/m^3$ ist nicht öfter als 18-mal im Jahr zulässig, woraus sich ein Jahrengrenzwert von $40\mu g/m^3$ ergibt. [Umw16]

Anhand dieser Kriterien ist das Potential des CEP in diesem Themenbereich erneut deutlich zu erkennen. Die zeitbasierten Eigenschaften dieser Vorgaben in Kombination mit den zu beobachteten Häufigkeiten bezogen auf ein Gebiet lassen sich optimal mit Hilfe von CEP-Regeln abbilden. Außerdem können leicht Änderungen an der Anwendung vorgenommen werden, falls sich die Vorgaben durch neue Richtlinien und Gesetzgeber ändern sollten, in dem die Regeln an diese angepasst werden. Dadurch erhält die Anwendung einen äußerst zeitresistenten Charakter.

4.4 Entwicklung des EPN-Netzwerks

4.4.1 Grundlegende Ereignisse

Auf Grundlage der festgehaltenen Kriterien zur Luftqualitätsanalyse, können wir die ersten einfachen Ereignisse bestimmen. Daraus werden wir ein erstes Ereignismodell erstellen, welches uns bei der Entwicklung des CEP-Netzwerks unterstützen wird. Aus den Verarbeitungsschritten in unserem Netzwerk werden sich neue Ereignisse und Ereignisquellen ableiten, welche wir dann in unser bisher erarbeitetes Ereignismodell aufnehmen.

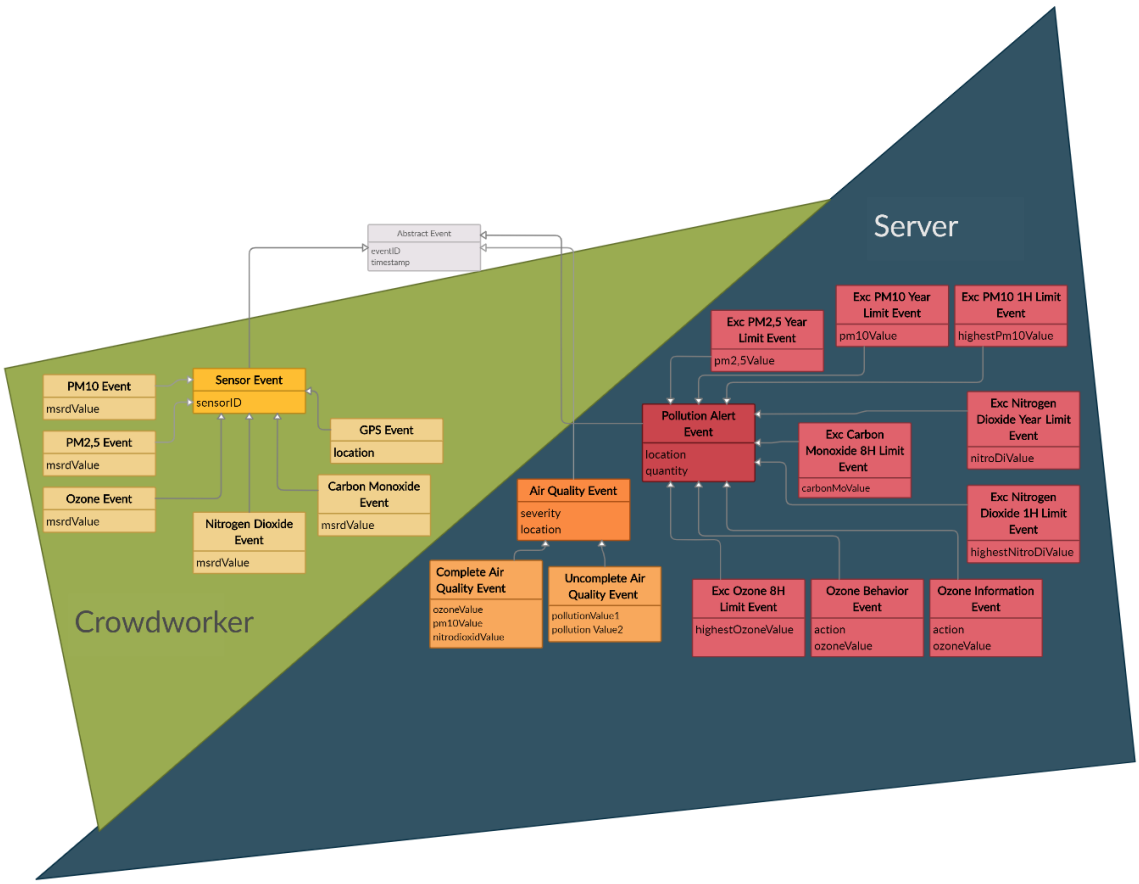

Abbildung 4.3: Erstes Ereignismodell ausgehend der detektierten Luftqualitätskriterien mit physischer Verteilung

Abbildung 4.3 zeigt eine erste Version unseres Ereignismodells. Zusehen sind die drei fachlichen Ereignistypen *Sensor Event*, *Air Quality Event* und *Pollition Alert Event*.

Sensor Event

Die Ausgangsereignisse in unserem Szenario stellen die *Sensor Events* dar. Über die am Raspberry Pi angeschlossenen Sensoren werden Daten über die verschiedenen Schadstoffe ermittelt. Dabei stellt jeder dieser gemessenen Schadstoffe ein eigenes primitives Ereignis dar. Diese Ereignisse müssen neben den gemessenen Werten auch Daten über Zeit und Herkunft der Messung enthalten. Um den Crowdworkern und damit den Luftdaten einen Standort zuordnen zu können, ermittelt ein GPS-Sensor die zugehörigen GPS-Daten, woraus Standort-Ereignisse resultieren.

Air Quality Event

Unsere Ausgangsereignisse werden eine Verarbeitungspipeline durchlaufen, an deren Ende einerseits die komplexen *Air Quality Events* ausgelöst werden. Diese Ereignisse repräsentieren die in Abschnitt 4.3 dargestellten Anforderungen an die Luftqualität im Sinne des LQI, welche sich aus der Aggregation und Verarbeitung der Sensor Events ergeben werden. Aus dem Air Quality Event leiten sich zwei Sub-Eventtypen ab:

- **Un-/Complete Air Quality Event**:
  Diese Ereignistypen stehen für den aus den Messwerten ermittelten konkreten Luftqualitätsindex. Das *Complete Air Quality Event* wird ausgelöst, wenn in dem jeweiligen Gebiet jeder der drei für die Berechnungsgrundlage notwendigen Schadstoffe PM_{10}, O_3 und NO_2 ermittelt werden konnte. Das *Uncomplete Air Quality Event* hingegen wird ausgelöst, sollten lediglich ein oder zwei Schadstoffwerte erfasst worden sein. Der letzendliche Wert des Luftqualitätsindex drückt sich durch das *severity*-Attribut aus, welches sich nach dem Wert mit der höchsten Konzentration richten.

Pollution Alert Event

Unseren zweiten Hauptereignistyp in dem Konzept stellen die *Pollution Alert Events* dar. Sie ergeben sich ebenfalls am Ende der Verarbeitungspipeline aus der Agggregation der Sensor Events. Aus ihnen leiten sich ebenfalls Sub-Eventtypen ab, die die verschiedenen Jahres- und Tagesgrenzwerte, sowie die Anzahl an zulässigen Überschreitungen repräsentieren. Auch drücken sich durch sie Handlungsempfehlungen aus, die bei erreichen bestimmter Ozonwerten nötig sind.

Neben den vorläufigen Ereignistypen ist diesem Ereignismodell ebenfalls eine physische Verteilung zu entnehmen. Die primitiven Sensor Events befinden sich dabei auf den von den Crowdworkern betriebenen Raspberry Pis, wo hingegen die komplexen Air Quality und Pollution Alert Events auf dem Server verarbeitet werden. Zwischen diesen beiden Ereignisdimensionen existieren jedoch noch weitere Abstraktionsstufen, welche sich im Durchlauf der Verarbeitungspipeline ergeben. Dazu müssen wir EPAs definieren, die

diese Verarbeitungsschritte übernehmen und uns letztlich die komplexen Ereignistypen liefern.

4.4.2 Das Event Processing Network

Um die letzte Abstraktionsstufe in unserem Ereignismodell zu erreichen, müssen wir im folgenden ein CEP-Netzwerk erarbeiten. Wie bereits in Abbildung 4.1 dargestellt, verteilen sich die EPAs in diesem Netzwerk dabei zum einen redundant auf die Raspberry Pis der Crowdworker. Deutlicher zeigt dies Abbildung 4.4, welche dieses EPN darstellt. Die Redundanz wird hier durch die mehreren Instanzen der jeweiligen Crowdworker EPA's deutlich. Außerdem verfügen einige dieser Crowdworker EPA's einen Zugriff auf sogenanntes *Area Context Knowledge*. Damit werden die eintreffenden GPS-Daten im späteren Verlauf mithilfe von zusätzlichen Diensten mit zusätzlich benötigtem Kontextwissen angereichert. Der andere Teil ist auf dem Server angesiedelt und für die Koordination der eingehenden Datenströme der Crowdworker zuständig. Überhaupt besitzen die verschiedenen EPAs verschiedene Aufgabenbereiche, die durch die in ihnen hinterlegen Regeln zur Mustererkennung klar umrissen werden. Außerdem wird durch die Regelverteilung die Komplexität unserer Anwendung reduziert und diese somit leichter verständlich.
Wir werden uns die einzelnen Verarbeitungsschritte anhand der dafür zuständigen EPAs und ihren Regeln anschauen. Die daraus abstrahierenden neuen Ereignistypen werden wir dann schließlich in unser Ereignismodell aufnehmen und es somit ergänzen.

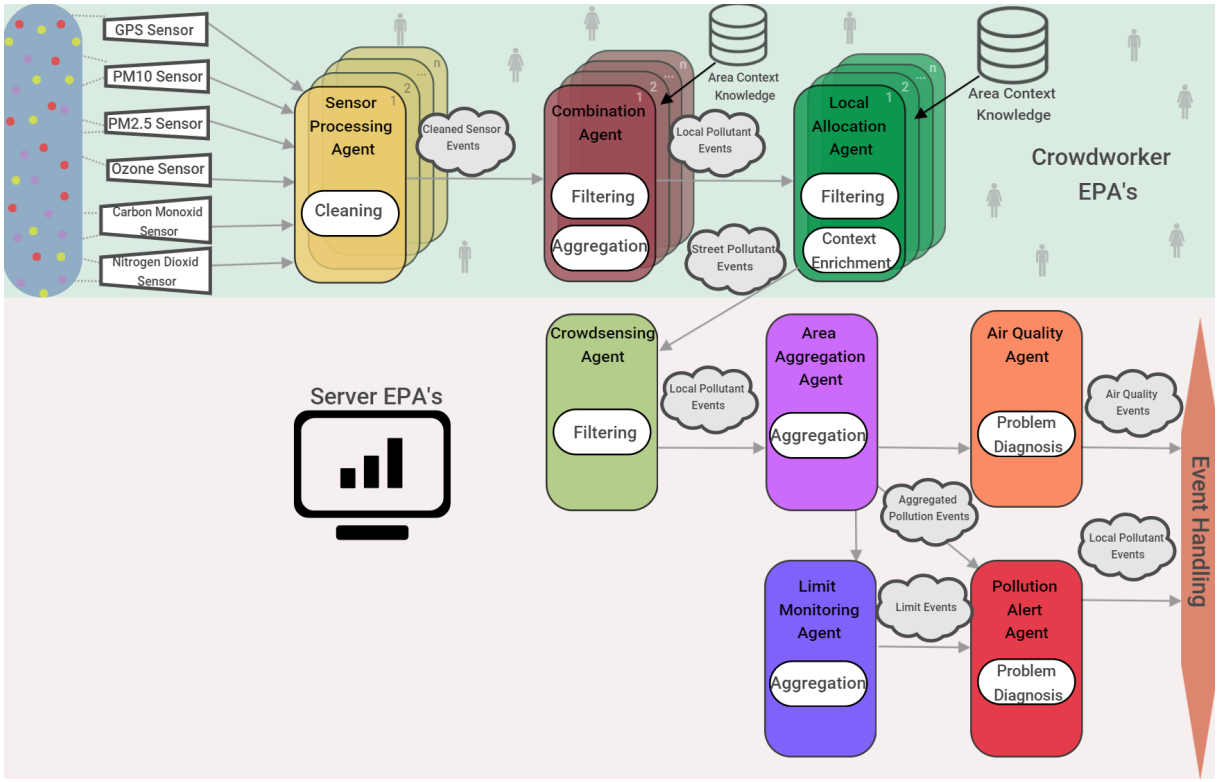

Abbildung 4.4: Darstellung der Verarbeitungs-Pipeline durch die EPAs im CEP-Netzwerk

Der Sensor Processing Agent

Unsere Hauptereignisquelle stellen die an die Raspberry Pis angeschlossenen Sensoren in unserem Konzept dar. Die durch sie gemessenen Daten bilden das Fundament der gesamten Anwendung und sind daher von signifikanter Relevanz. Bei der Datenerhebung mit solchen Sensoren kann es jedoch vorkommen, dass inkonsistente Werte auftreten die nicht in einen realistischen Rahmen fallen. Der *Sensor Processing Agent* dient deshalb zur einfachen Datenbereinigung und stellt die Konsistenz der ermittelten Messwerte sicher. Bei GPS-Daten kann es beispielsweise zu unrealistischen Sprüngen kommen, die wir nicht in unsere Verarbeitung miteinbeziehen wollen. Auch unsere Schadstoffsensoren können Daten liefern, die für uns nicht zu gebrauchen sind. Das sind vor allem Werte im negativen, oder ungewöhnlich hohem Bereich. Zur Umsetzung definieren wir uns eine einfach Regel:

CONDITION: (every PM10Event as p)
$\wedge$ (p.msrdValue $>$ 0 $\wedge$ p.msrdValue $<$ 200)

ACTION: create CleanedPM10Event(msrdValue = p.msrdValue)

Die Regeldefinitionen für die anderen Schadstoffwerte erfolgen analog. Für die Bereinigung der GPS-Daten machen wir uns den Geschwindigkeitswert zunutze, der uns durch den GPS-Sensor ebenfalls übermittelt wird. Bei zu hohen Geschwindigkeiten wird es schwierig, den räumlichen und semantischen Bezug der Daten konsistent zu halten. Auch ist dabei eine korrekte Funktionsweise der Sensoren nicht gewährleistet, wie bereits in Abschnitt 4.2 erwähnt. Deshalb leiten wir nur GPS-Daten weiter, die bei Geschwindigkeiten von unter 10km/h ermittelt wurden. So stellen wir sich, dass lediglich Messungen für die weitere Verarbeitung verwendet werden, die nicht im Auto oder öffentlichen Verkehrsmittel gemessen wurden. Die daraus entstehenden Bereinigungsereignisse bilden die erste Ergänzung für unser Ereignismodell, wie in Abbildung 4.5 dargestellt.

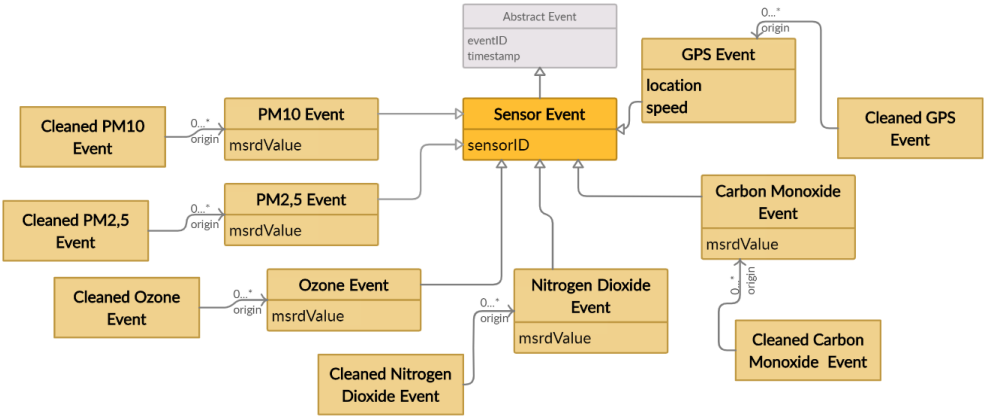

Abbildung 4.5: Bereinigte Sensorereignisse des Sensor Processing Agents

Der Combination Agent

Der ausschlaggebende Grund für die Weiterverarbeitung der Sensordaten ist, dass sie von feingranularer Natur sind und somit wenig Aussagekraft besitzen. Aktuell haben wir nur alleinstehende, bereinigte Ereignisse, welche uns so noch wenig über die beobachtete Situation aussagen können. Deshalb müssen wir die Messwerte in einem räumlichen Bezug setzen. Ermöglicht wird dies mit dem *Combination Agent*, der diese Messwerte mit ihren zugehörigen GPS-Daten in einem neuen Ereignistyp vereint. Dafür betrachten wir jedes bereinigte Schadstoffereignis einzeln und kombinieren es mit dem zugehörigen Standortereignis. Nachfolgend definieren wir uns dazu eine Regel:

CONDITION: (CleanedPM10Event as p
→ CleanedGPSEvent as g)[win:time:3sec]
∧ (MapService.inBuilding(g.location) = 'FALSE')

ACTION: create LocalPM10Event(msrdValue = p.msrdValue, location = g.location, speed = g.speed)

Die hier beschriebene Regel erwartet innerhalb von 3 Sekunden nach Eintreffen eines CleanedPM10Events ein CleanedGPSEvent. Durch die vorangegangene Geschwindigkeitsbeschränkung auf < 10km/h in Kombination mit dem betrachteten Zeitfenster, beschränken wir den zuzuordnenden Radius des Messwertes auf 5-10 Meter. Außerdem bedienen wir uns hier das erste Mal dem Area Context Knowledge. Anhand der GPS-Daten überprüfen wir mit einem Kartendienst, ob die empfangenen Koordinaten in einem Gebäude liegen, oder einem Gebiet außerhalb eines Gebäudes zuzuordnen sind. Ist letzteres der Fall, werden die Attribute dieser aufeinanderfolgenden Werte in einem LocalPM10Event zusammengefasst und an die nächste Verarbeitungsstufe weitergeleitet. Die Regeln für die anderen Schadstoffe erfolgen wieder entsprechend der hier beschriebenen. Unser Ereignismodell reichern wir nun mit diesen neu hinzugekommenden, in Abbildung 4.6 dargestellt Ereignissen an.

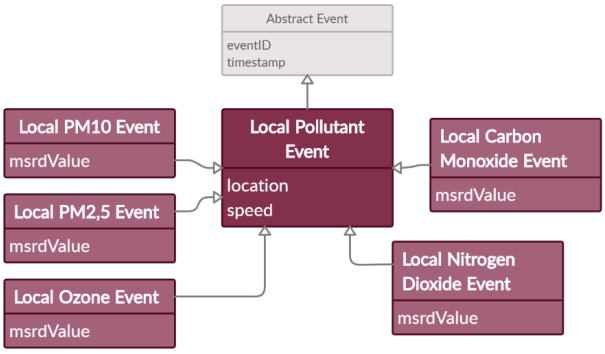

Abbildung 4.6: Kombinierte GPS- und Sensorereignisse des Combination Agents

Der Local Allocation Agent

Anhand der GPS-Daten, welche sich durch die Latitude- und Longitudewerte unserer Ereignisse ausdrücken, können wir noch keine räumliche Auswertung durchführen. Wir benötigen weiteres Kontextwissen des Area Context Knowledge, um diese Standortdaten in für uns fachlich verständliche Informationen umzuwandeln. Eine Möglichkeit wäre, eine Art Raster über das zu untersuchende Gebiet zu legen, welches die jeweiligen GPS-Daten in einem übergeordneten Rasterabschnitt zusammenfasst. Eine andere Möglichkeit ist das *Reverse Geocoding*, welches wir auch in der prototypischen Umsetzung verwenden werde. Dabei werden Standortdaten in für uns lesbare Adressdaten umgewandelt. Wir können uns so an der bestehenden Infrastruktur orientieren und die Daten für einzelne Straßen zusammenfassen und auswerten. Diesen Kontextzugriff übernimmt der *Local Allocation Agent.*
Neben der räumlichen Zuordnung ist jedoch ebenfalls noch ein Filterschritt nötig. So kann es nämlich vorkommen, dass kontinuierlich Ereignisse mit den gleichen Attributen eintreffen und damit aktuell keine Veränderungen in der Anwendungsdomäne beobachtet werden. Wenn sich unser Crowdworker also über längere Zeit nicht bewegt und die jeweiligen erfassten Schadstoffe währenddessen keine unterschiedlichen Werte annehmen, wollen wir diese eintreffenden Daten als für uns nicht relevant ansehen und rausfiltern. In der folgenden Regel fassen wir diesen Filterschritt mit der Wissensanreicherung zur räumlichen Zuordnung zusammen:

CONDITION: (LocalPM10Event as l1
$\rightarrow$ LocalPM10Event as l2)[win:length:2]
$\wedge$ ((l1.speed = 0 $\wedge$ l2.speed=0 $\wedge$ l1.msrdValue $\neq$ l2.msrdValue)
$\vee$ (l1.speed = 0 $\wedge$ l2.speed $>$ 0)
$\vee$ (l1.speed $>$ 0 $\wedge$ l2.speed = 0)
$\vee$ (l1.speed $>$ 0 $\wedge$ l2.speed $>$ 0))

ACTION: create PM10StreetEvent(msrdValue = l1.msrdValue, street = RevGeocodingService.getAdress(l1.location))

Wir betrachten in dieser Regel immer zwei aufeinanderfolgende Ereignisse des Typs LocalPM10Event. Da wir bestimmte Ereignisse nicht einfach aus dem Ereignisstrom entfernen können, decken wir in unserer Regel alle Fälle ab, die sich von gleichbleibenden Messwerten an einem Standort unterscheiden. Sobald also ein zweites LocalPM10Event eintrifft, wird anhand der Geschwindigkeiten geprüft, ob die Messwerte demselben GPS-Punkt zuzuordnen sind. Falls ja, überprüfen wir die Verschiedenheit der Schadstoffwerte. Falls nein, müssen wir nur sicherstellen, dass unser Crowdworker die Messwerte an zwei verschiedenen GPS-Punkten ermittelt hat. Sobald eine der Bedingungen feuert, erzeugen wir uns ein neues PM10StreetEvent. Dieser Ereignistyp übernimmt den Schadstoffwert des zuerst eingetroffenen Ereignisses und ermittelt über den Zugriff auf eine Serviceanwendung den Straßennamen anhand der Positionsdaten. Mit diesen Ereignistypen lie-

gen uns zwar immer noch alleinstehende Werte vor, jedoch besitzen sie aufgrund der Kontextanreicherung einen für uns sinnvollen räumlichen Bezug. Dargestellt sind diese Ereignistypen in Abbildung 4.7. Diese Ereignisse werden nun von jedem eizelnen Crowdworker mittels TCP-Verbindung zur Auswertung an den Server gesendet.

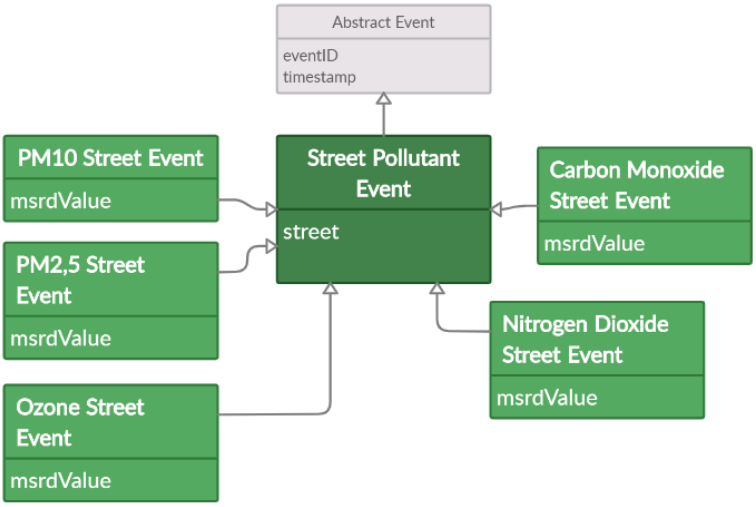

Abbildung 4.7: Angereicherte Ereignisse des Local Allocation Agents

Der Crowdsensing Agent

Die bisher betrachteten EPAs haben die Vorverarbeitung der Datenströmen auf den Rasperry Pis der Crowdworker übernommen. Dabei wurden die rohen Sensordaten in höherwertige Ereignistypen synthetisiert. Um diese in der Crowd verteilten Ereignisströme nun zusammenzufassen, müssen wir sie an zentraler Stelle zusammenfließen lassen, koordinieren und auswerten. Einstiegspunkt dabei ist der *Crowdsensing Agent*. Er ist auf dem Server angesiedelt und empfängt die gesamten Ereignisse der einzelnen Crowdworker. Daher nimmt er eine äußerst wichtige Rolle in unserem Konzept ein.

Trotz der Vorfilterung auf den Raspberry Pis, ergibt sich für den Server eine riesige Datenlast, die es zu verarbeiten gilt. Sie resultiert aus der Menge der einzelnen Crowdworker-Datenströme und bedarf einer weiteren Filterung zur Ereignisreduzierung. Dieser Filterschritt gestaltet sich jedoch etwas anders als auf den einzelnen Raspberry Pis. Die Wahrscheinlichkeit, dass in den einzelnen eintreffenden Ereignissen Duplikate vorkommen, ist hier aufgrund der Unabhängingkeit und Vielzahl der Crowdworker sehr hoch. Gleichzeitig treffen kontinuierlich Werte aus verschiedenen Straßen ein. Wir müssen also pro Schadstoff und beobachteter Straße individuelle Werte detektieren. Da wir dies jedoch nicht über den gesamten Beobachtungszeitraum tun können, müssen wir uns auf eine periodische Filterung der doppelten Werte beschränken. Das Vorgehen des Crowdsensing Agents ist in Abbildung 4.8 dargestellt.

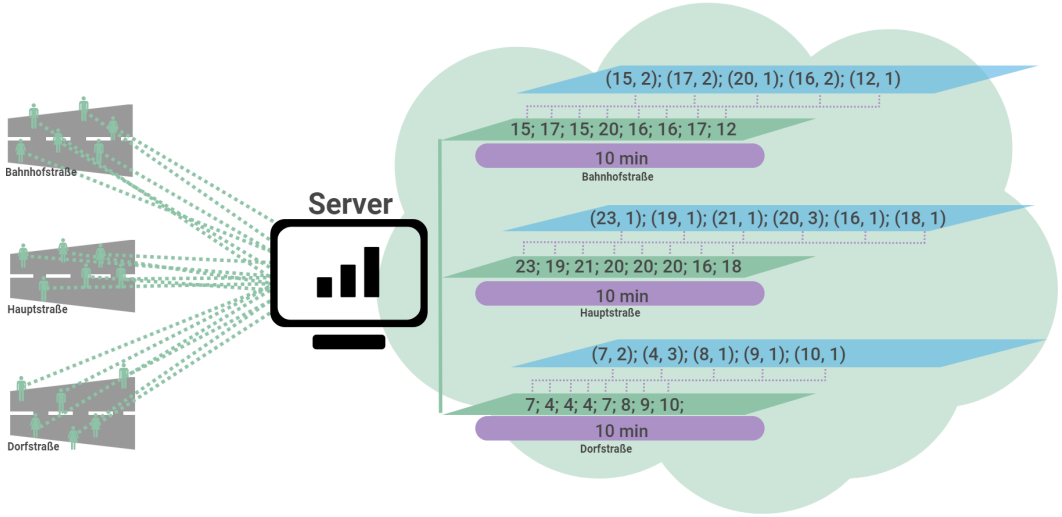

Abbildung 4.8: Filterung der Crowdsensingwerte pro Straße durch den Crowdsensing Agent

Wir wählen ein Zeitfenster von 10 Minuten. In diesem Zeitraum betrachten wir pro beobachtete Straße die eintreffenden Werte gesondert und fassen sie anhand der Häufigkeit ihres Auftretens in einem neuen Zwischen-Ereignistyp zusammen. Wenn wir die Duplikate einfach aus dem Ereignisstrom entfernen, würden wir die nachfolgende Durchschnittsberechnung verfälschen und damit möglicherweise keine korrekten Erkenntnisse über den untersuchten Kontext erhalten. Aus diesem Grund erfolgt die Ereignisreduzierung auf diese weise. Dieser Schritt wird für jeden Schadstoff durchgeführt, woraus sich letztlich verwertbare Daten der Crowdworker in Form von Crowdsensed Events ergeben.

Bis hier hin ist viel an Vorverarbeitung geschehen. Im folgenden geht es um die wirkliche Verarbeitung der Daten hinsichtlich der Informationsgewinnung. Die Ereignistypen, von welchen dabei ausgegangen wird sind in Abbildung 4.9 dargestellt.

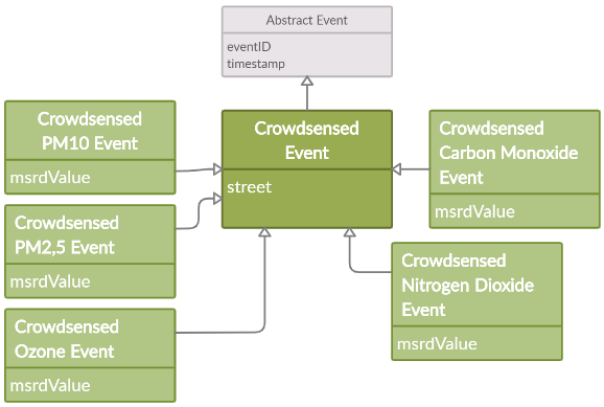

Abbildung 4.9: Gefilterte Crowdereignisse des Crowdsensing Agent

Der Area Aggregation Agent

Wir haben unsere Sensorereignisse nun soweit abstrahiert, dass wir diese nun mittels Aggregationsfunktionen räumlich sowie zeitlich Korrelieren können. Dafür verwenden wir den *Area Aggregation Agent*. Aus den gefilterten, durch die Crowd ermittelten Werte werden mit diesem Agenten Durchschnittswerte für jeden einzelnen Schadstoff auf jeder beobachteten Straße berechnet. Dabei orientieren wir uns bei der Regeldefinition an die in Abschnitt 4.3 beschriebenen Anforderungen an die Luftqualität. Demnach benötigen wir zur Ermittlung des Luftqualitätsindex stündliche Durchschnittswerte der drei Schadstoffe PM_{10}, Ozon und Stickstoffdioxid, welche pro Straße gebildet werden müssen. Die folgende Regel zeigt eine solche Durchschnittsbildung beispielhaft für eine spezifische Straße:

CONDITION: (CrowdsensedOzoneEvent as c)[win:time:batch:60min]
GROUP BY: street

ACTION: create 1HourAvgOzoneEvent(areaValue = avg(c.msrdValue), street =c.street)

Da wir zu jeder Stunde einen Mittelwert benötigen, verwenden wir ein *Batch-Window*. Dieses verhindert, dass nach jedem eintreffenden Ereignis eine neue Berechnung angestoßen und die Durchschnittsberechnung erst am Ende des Zeitfensters durchgeführt wird. Selbstverständlich können wir nicht einfach Regeln für jede mögliche Straße definieren, da wir zum einen nicht wissen welche Straßen von den Crowdworkern bewandert werden und es zum anderen schlicht nicht möglich ist, jede Straße so gesondert zu berücksichtigen. Wir würden eine nahezu endlose Anzahl an Regeln dafür benötigen. Deshalb verwenden wir hier die aus Abschnitt 3.3.1 bekannte Gruppierungsfunktion. Die sich aus den Durchschnittswerten ergebenden Ereignisse zeigt Abbildung 4.10.

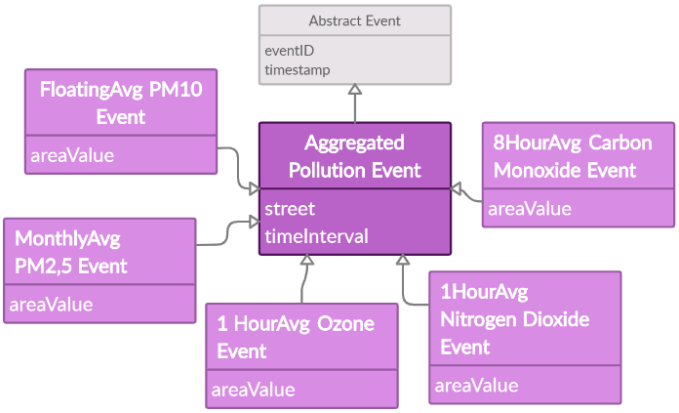

Abbildung 4.10: Durchschnittsereignisse der Schadstoffe des Area Aggregation Agent

Der Air Quality Agent

Die nun erhaltenen stündlichen Durchschnittswerte können wir nutzen, um den LQI zu berechnen. Dazu benötigen wir einen *Air Qualiry Agent*, der die berechneten Werte entgegennimmt und anhand dessen den LQI in der jeweiligen Straße bestimmt. Folgende Regel liefert uns einen vollständigen LQI mit der Bewertung *mäßig - gelb*:

CONDITION: (1HourAvgOzoneEvent as o
∧ FloatingPM10Event as p
∧ 1HourNitrogenDioxideEvent as n)[win:time:batch:60min]
∧ ((120 < o.areaValue < 181 ∧ p.areaValue < 36 ∧ n.areaValue < 41)
∨ (40 < n.areaValue < 101 ∧ p.areaValue < 36 ∧ o.areaValue < 120)
∨ (35 < p.areaValue < 51 ∧ n.areaValue < 41 ∧ o.areValue < 120))
GROUP BY: street

ACTION: create CompleteAirQualityEvent(
severity = ”MODERATE”,
street = o.street,
ozoneValue = o.areaValue, pm10Value = p.areaValue, nitrodioxidValue = n.areaValue)

Wir benötigen Durchschnittswerte von PM_{10}, Ozon und Stickstoffdioxid. Das Muster feuert, sobald einer der Werte in seinem spezifischen *mäßigen* Bereich liegt (siehe Tabelle in Abschnitt 4.3). Die anderen Werte dürfen dabei nicht über ihren jeweiligen Wert im Bereich *gut* liegen. Um stündliche Aussagen über den LQI treffen zu können, definieren wir erneut ein Batch-Window von einer Stunde. Es wird dann ein neues Complete Air Quality Event erzeugt, das Daten über die Bewertung, den Standort sowie die Zusammensetzung durch die Durchschnittswerte beinhaltet. Die Bewertung findet dabei erneut mittels Gruppierung anhand der Straßennamen statt.

Für den Fall, dass nicht alle Werte vorhanden sind, werden Uncomplete Air Quality Events erzeugt. Die Regeldefinition gestaltet sich relativ ähnlich. Dabei unterscheiden wir zwischen einem unvollständigen LQI mit einem und mit zwei Werten. Für ersteres werden Regeln benötigt, die die einzelnen Schadstoffdurchschnittswerte pro Bewertungsstufe gesondert betrachten. Letzteres hingegen benötigt Regeln, die ebenfalls pro Bewertungsstufe die verschiedenen Paarmöglichkeiten der Schadstoffe berücksichtigt.

Mit den uns nun vorliegenden komplexen Ereignissen haben wir einen Teil der höchsten Abstraktionsstufe in unserem Konzept erreicht. Wir können diese Ereignisse nun nutzen, um Aussagen über die Luftqualität auf verschiedenen Straßen zu treffen und beispielsweise eine Übersichtskarte erstellen. Zum Abschluss unseres konzeptionellen Entwurfs fehlt jedoch noch ein Teil in unserer Verarbeitungspipeline.

Der Limit Monitoring Agent

Neben dem LQI haben die einzelnen Schadstoffe noch Tages- und Jahresgrenzwerte (wie in Abschnitt 4.3 beschrieben), die wir ebenfalls beobachten wollen und dazu erst auswerten müssen. Der *Limit Monitoring Agent* nimmt sich dafür die voraggregierten Werte des Area Aggregation Agent und bildet daraus erneut Durchschnittswerte über verschiedene Zeiträume. Im folgenden formulieren wir eine Regeln zur Überwachung des $PM_{2,5}$-Jahresgrenzwerts:

> CONDITION: (MonthlyAvhPM2.5Event as m)[win:time:batch:365days]
> GROUP BY: street
>
> ACTION: create PM2.5YearLimitEvent(areaValue = avg(m.areaValue), street =m.street)

Die reine Funktion dieses Agenten unterscheidet sich zu der des Area Aggregation Agents nicht. Dennoch entstehen hier weitere, neue Ereignistypen, die wir in unser Ereignismodell aufnehmen müssen und in Abbildung 4.11 dargestellt sind. Das nachfolgende Grenzwertmonitoring erfolgt dann anhand dieser Ereignisse.

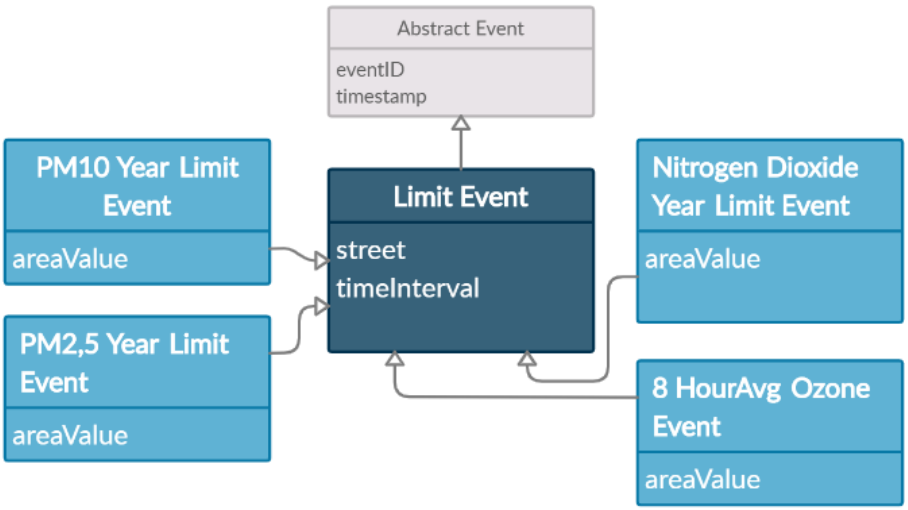

Abbildung 4.11: Grenzwertereignisse der Schadstoffe des Limit Monitoring Agent

Der Pollution Alert Agent

Nach dem wir nun den LQI bestimmt und verschiedenen Durchschnittswerte zur Luftanalyse berechnet haben, müssen wir in einem letzten Schritt überprüfen, ob die verschiedenen Grenzwerte gemäß ihrer Richtlinen eingehalten werden. Dazu benötigen wir einen letzten Agenten, der die verschiedenen Grenzwerte entgegen nimmt und sie anhand ihrer Anforderungen analysiert. Wir definieren also einen *Pollution Alert Agent*, welcher zum einen die produzierten Ereignisse des Area Aggregation Agents zur stündlichen Grenzwertkontrolle verarbeitet. Dafür definieren wir erneute eine beispielhafte Regel für die Überwachung des Ozon-Grenzwerts:

CONDITION: (1HourAvgOzoneEvent as o)
$\wedge$ (o.areaValue $>$ 179)
GROUP BY: street

ACTION: create OzoneInformationEvent(
action = "Recommendation of behavior to the population necessary",
street = o.street,
ozoneValue = o.areaValue)

Sollte der stündliche Ozon-Durchschnittswert über $179\mu g/m^3$ liegen, sind die Behörden dazu verpflichtet die Bevölkerung über diesen Zustand zum Schutz der Gesundheit zu informieren. Bei Werten ab $240\mu g/m^3$ ist gar eine Verhaltensempfehlung auszusprechen. Eine Regel würde dementsprechend analog formuliert werden. Das hier erzeugte *Ozone Information Event* lief ert uns diese Information und kann nun beispielsweise einen Geschäftsprozess antoßen, der genau diese Aktion ausführt.

Andererseits dürfen einige Grenzwerte mehrmals im Jahr überschritten werden. Für diese müssen wir die Anzahl an festgestellten Überschreitungen festhalten. Ist die zulässige Anzahl erreicht, erzeugt der Agent ein Überschreitungsereignis, welches in einem weiteren Schritt ein Warnereignis auslöst. Dieses kann dann wie schon vorhin genutzt werden, um nachgelagerte Systeme anzusprechen und entsprechende Maßnahmen zu treffen. Die Regeldefinition dazu folgt in Abbildung 6.1.

4.4.3 Abschließendes Ereignismodell

Die anfänglich primitiven Sensorereignisse sind nun über mehrere Verarbeitungsschritte hinweg zu komplexen Air Quality und Pollution Alert Events synthetisiert worden. Dabei sind verschiedene Ereignistypen entstanden, die wir schließlich in unser Ereignismodell aufnehmen müssen, um es zu ergänzen. In Abbildung 4.13 und ?? ist dieses vollständige Ereignismodell zusammen mit den Strukturen der verschiedenen Ereignistypen und den Beziehungen zwischen ihnen dargestellt.

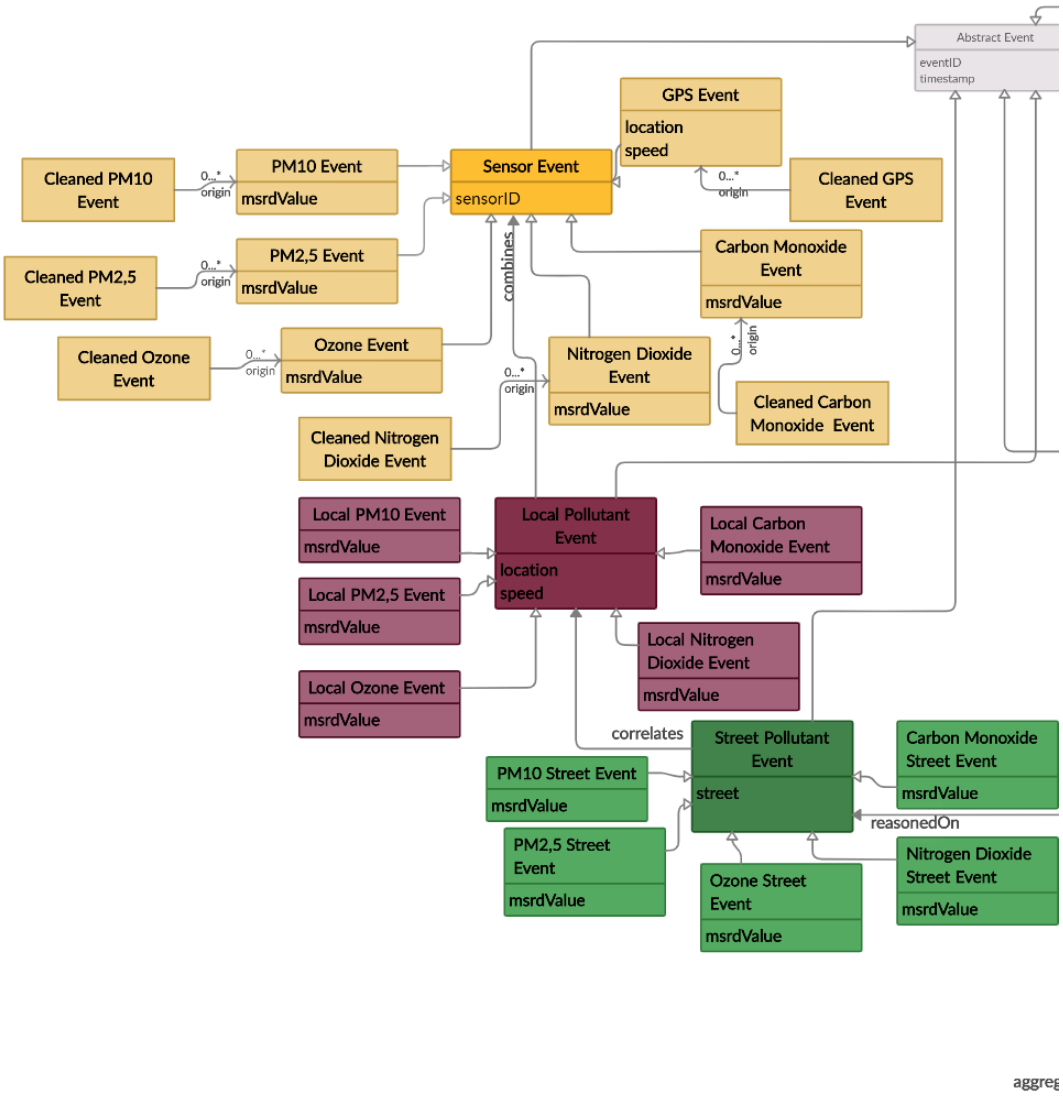

Abbildung 4.12: Vollständiges Ereignismodell (1)

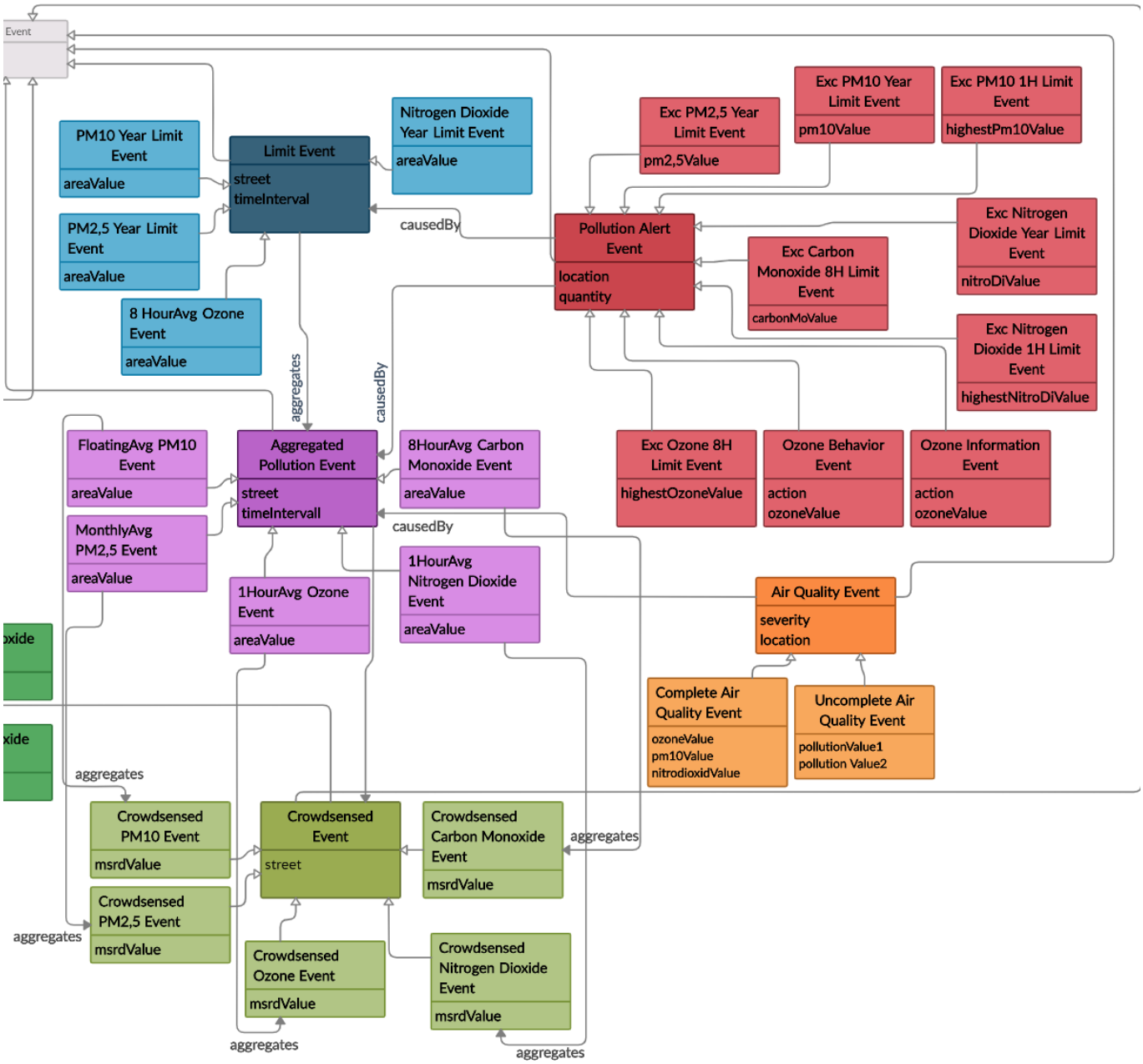

Abbildung 4.13: Vollständiges Ereignismodell (2)

5 Prototypische Umsetzung des Konzepts

Wir haben ein Konzept entwickelt, welches nun in gewisser Weise umgesetzt werden muss. Dabei beschränken wir uns jedoch auf die Implementierung der Anwendung auf einem Raspberry Pi, sowie der zentralen Verarbeitungskomponente. Damit soll gezeigt werden, dass die einzelnen Komponenten des Konzepts wie erwartet miteinander interagieren und eine Umsetzung in der Realität rein technisch gesehen möglich ist. Dies umfasst die Ansteuerung der Sensoren auf dem Raspberry Pi, die Verarbeitung der erfassten Daten dieser Sensoren sowie die Verwendung des CEP auf Rasbperry Pi und Server mit der CEP-Engine *Siddhi*.

5.1 Die CEP-Engine Siddhi

5.1.1 Einführung

Da wir mit dem Raspberry Pi keinen vollwertigen Heimrechner haben, lag der Anspruch bei einer leichtgewichtigen CEP-Engine, die auch auf diesem Gerät zur Verarbeitung der erfassten Sensorwerte verwendet werden kann. Gleichzeitig muss sie fähig sein, als Server zu fungieren und mit der durch die Crowdworker produzierten Datenlast fertig zu werden. Mit der Open-Source CEP-Engine Siddhi haben wir eine leistungsstarke Alternative gefunden, die diesen Anforderungen gerecht wird und deshalb in dieser Arbeit verwendet wird.

Siddhi, ursprünglich ein Forschungsprojekt, wird aktuell von der WSO2 Inc. unter der Apache License Version 2 bereitgestellt und ist unter anderem bei namenhaften Unternehmen wie PayPal, eBay und Uber im Einsatz. Letzteres veröffentlichte Zahlen, laut denen sie täglich über 20 Milliarden Ereignisse mit Siddhi zur Betrugserkennung verarbeiten, was ihre Leistungsfähigkeit noch einmal unterstreicht. [Vis17] [Inc20b]

Wir werden Siddhi mit der Version 5.1 in dieser Arbeit eingebettet als Java-Bibliothek verwenden. Abseits davon kann die Engine genau so in Python eingebunden werden oder als eigenständiger Microservice agieren. Im folgenden schauen wir uns die Architektur der Engine mit ihren einzelnen Komponenten an, um den Ablauf während der Ereignisverarbeitung innerhalb der Engine nachvollziehen zu können. Anschließend wird

demonstriert, wie eine Siddhi Anwendung in Java implementiert wird. Dazu wird die hier verwendete Ereignisanfragesprache *Siddhi Streaming SQL* anhand eines Beispiels präsentiert, um die Regeldefinitionen in der nachfolgenden Umsetzung des Konzepts verstehen zu können.

5.1.2 Architektur der Engine

Die Architektur der Engine orientiert sich im wesentlichen nach dem üblichen Vorgehen im Complex Event Processing. In Abbildung 5.1 sind diese wesentlichen Strukturen innerhalb einer Siddhi Anwendung dargestellt. Sources und Sinks stellen dabei den Eingang bzw. Ausgang des Datenstroms dar. Die durch die Ereignisquellen produzierten Daten treffen in ihren unterschiedlichen Datenformaten hier ein und werden nach ihrer Verarbeitung an derer Stelle wieder veröffentlicht. [Inc20a] Für diese Bearbeitung werden jedoch verschiedenen Komponenten benötigt.

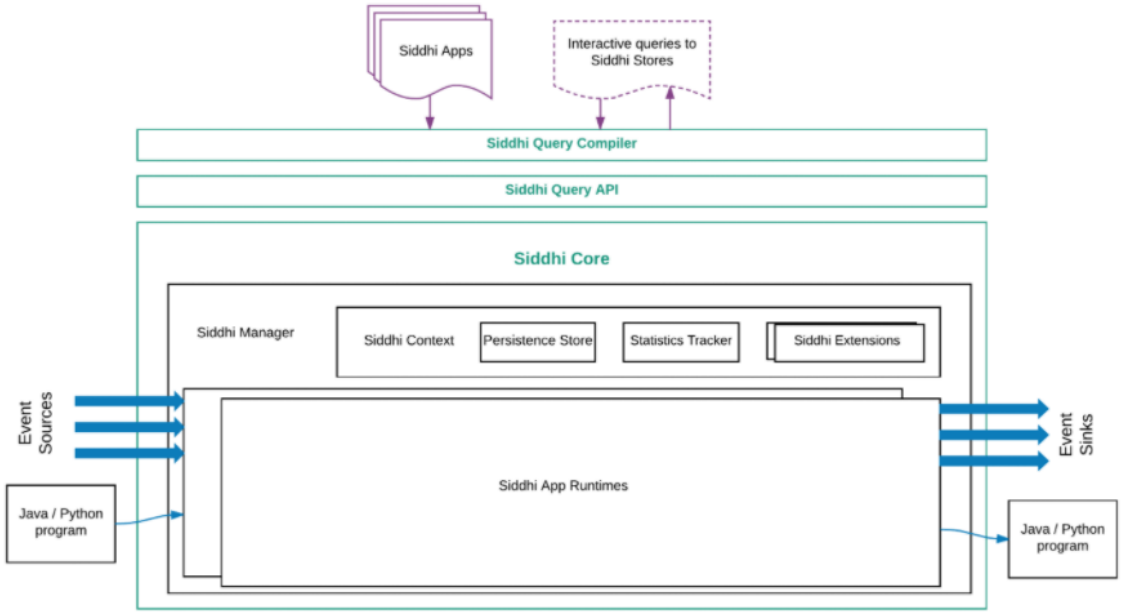

Abbildung 5.1: Architektur der Siddhi Engine mit ihren Hauptkomponenten, vgl. [Inc20a]

Siddhi Application (Apps)

Eine Siddhi Application beschreibt die Ausführungslogik der Engine, in dem durch sie Regeln zur gewünschten Ereignisverarbeitung definiert werden. [Inc20a]

Siddhi App Runtimes

Wie auch mit der JRE in Java werden durch die Siddhi App Runtimes seperate Laufzeitumgebung zur Ausführung der Siddhi Apps bereitgestellt. Dabei sind mehrere Instanzen möglich, welche sich jedoch nach den in der Ausführungslogik beschriebenen Anwendungsfällen richten sollte. Sie repräsentieren quasi die Event Processing Agents in einem EPN. [Inc20a]

Siddhi Manager

Diese Komponente verwaltet die Siddhi App Runtimes mithilfe der unter Siddhi Context bereitgestellten Zugriffsmöglichkeiten und Erweiterungen. Entgegen den Siddhi App Runtimes existiert nur ein Siddhi Manager innerhalb einer Anwendung. [Inc20a]

Für die Kommunikation mit den Sources und Sinks ist das **Siddhi Core** Modul zuständig. Es ist zentraler Bestandteil der Engine führt unter Zuhilfenahme des Siddhi Managers und der Siddhi App Runtimes die Ereignisverarbeitung durch, weshalb sie auch in diesem Modul angesiedelt sind. Zur Verarbeitung der Siddhi Apps ist ein **Siddhi Query Compiler** nötig. Dieser übersetzt die in den Siddhi Apps definierten Regeln in sogenannte Siddhi Query API POJOS. Zur Kommunikation zwischen dem Siddhi Core und Query Compiler Modul, nutzt die **Siddhi Query API** die übersetzten POJO-Objekte und ihre bereitgestellten Methoden und fungiert damit als Bindeglied zwischen diesen Modulen. [Inc20a]

5.1.3 Implementierung einer Siddhi-Anwendung in Java

Um Siddhi in einer Java-Anwendung implementieren zu können wird, Maven benötigt. Dazu müssen zunächst die nötigen Abhängigkeiten in der Maven üblichen pom.xml Konfigurationsdatei eingebunden werden. Das ist alles an nötiger Vorarbeit und es kann zum Code übergehen. Dazu betrachten wir das in Abbildung 5.2 implementierte Beispiel eingehender Feinstaubereignisse.

Als erstes wird unter Verwendung der Siddhi Streaming SQL eine Siddhi App definiert (**Zeile 1-5**). Wie der Name bereits vermuten lässt, orientiert sich diese EPL stark an der Datenbank-Abfragesprache SQL. In dieser Siddhi App wird ein Eingangsstrom namens *PM10Stream* mit den Attributen *symbol* und *msrdVal* definiert. Das besondere an Siddhi ist, dass die Definition der Ereignistypen impliziert durch den Ereignisstrom bestimmt wird. In **Zeile 3** beginnt die Definition der eigentlichen Ereignisregel. Die Regel legt ein Batch-Window der Länge 3 fest. Sobald drei Ereignisse aus dem vorher definierten Ereignisstrom eingetroffen sind, wird der Durchschnittswert der drei eingetroffenen PM10Events berechnet. Um die Ergebnisse auswerten zu können, werden sie schließlich in einen Ausgabestrom umgeleitet (**Zeile 5**).

Um die Regel nun ausführen zu können, wird ein Siddhi Manager benötigt. Dieser erzeugt die Laufzeitumgebung Siddhi App Runtime, welche wiederum die Siddhi App zur Ausführung übergeben bekommt (**Zeile 7-8**).

Für das Event-Handling kann ein StreamCallBack definiert werden (**Zeile 10-16**). Dieser ermöglicht es, die eingetroffenen und verarbeiteten Events eines bestimmten Streams an nachgelagerte Systeme weiterzuleiten und so weitere Geschäftsprozesse anzustoßen. In unserem Beispiel lassen wir die Ergebnisse des AggregateStockStream lediglich ausgeben.

```
1   String siddhiApp = "" +
2               "define stream PM10Stream (symbol string, msrdVal float); " +
3               "from PM10Stream#window.lengthBatch(3) " +
4               "select symbol, avg(msrdVal) as avgVal " +
5               "insert into AvgPM10Stream ;";
6
7       SiddhiManager siddhiManager = new SiddhiManager();
8       SiddhiAppRuntime siddhiAppRuntime = siddhiManager.createSiddhiAppRuntime(siddhiApp);
9
10      siddhiAppRuntime.addCallback("AvgPM10", new StreamCallback() {
11          @Override
12          public void receive(Event[] events) {
13              System.out.println("Average Event: ");
14              EventPrinter.print(events);
15          }
16      });
17
18      InputHandler inputHandler = siddhiAppRuntime.getInputHandler("PM10Stream");
19
20        siddhiAppRuntime.start();
21
22      inputHandler.send(new Object[]{"PM10", 40f});
23      inputHandler.send(new Object[]{"PM10", 30f});
24      inputHandler.send(new Object[]{"PM10", 80f});
25      inputHandler.send(new Object[]{"PM10", 20f});
```

Abbildung 5.2: Beispielhafte Auswertung von Aktienpreisen und -handelsvolumina mit einer Siddhi-Anwendung in Java

Um die Daten jedoch überhaupt im Eingangsstrom erfassen zu können, ist ein InputHandler notwendig (**Zeile 18**). Dieser erzeugt aus den Daten der jeweiligen Ereignisquelle Ereignisse und leitet sie in den Eingangsstrom um.

Als letztes muss die Laufzeitumgebung noch gestartet werden, um die Ereignisverarbeitung zu beginnen (**Zeile 20**).

In **Zeile 22-25** ist noch zusehen, wie dem Eingangsstrom über den InputHandler manuel Ereignisse gesendet werden können. In dem Beispiel sind dies vier Ereignisse. Die

Regel in **Zeile 3** jedoch betrachtet ein Längenfenster der Größe drei. Somit wird nach Eintreffen des dritten Ereignisses (**Zeile 24**) die Berechnung durchgeführt und das vierte Ereignis (**Zeile 25**) in die nächste Berechnung mit einbezogen.

Das hier behandelte Beispiel dient lediglich zum grundlegenden Verständnis der Siddhi Streaming SQL. Die Regeldefinitionen in der nachfolgenden Implementierung des Konzept gehen jedoch über die hier beschriebenen Anfragemethoden hinaus. Sollten Probleme bei der Nachvollziehbarkeit der Regeln auftreten, sollte die Dokumentation der Siddhi Streaming SQL unter [Inc20c] herangezogen werden.

5.2 Implementierung der Anwendung

5.2.1 Die Crowdworker Anwendung

Hardware

Wie bereits mehrfach erwähnt, benötigen wir zunächst einen Raspberry Pi (Abschnitt 2.2). Die Voraussetzungen für die Inbetriebnahme wurden bereits zum Teil in Abschnitt 4.2 erwähnt (Stromversorgung, Netzwerkverbindung). Vorher muss jedoch das aktuelle Betriebsystem auf dem Gerät installiert werden. Dazu ist eine SD-Karte nötig, auf die das OS-Image aufgespielt werden muss. Um die höchste Performance des Geräts zu erhalten, wurde in dieser Arbeit das *Raspberry Pi OS (32-bit) Lite* verwendet. Dabei handelt es sich um eine kostenlos erhältliche Minimalversion des offiziellen für den Raspberry Pi vertriebenen Betriebssystems ohne Desktop-Umgebung. [Fou20d] Aufgrund der fehlenden Benutzeroberfläche ist es ebenfalls nötig, auf dem Gerät eine SSH-Verbindung einzurichten, mit der auf das Gerät zugegriffen werden kann. Sobald auf dem Gerät alle nötigen Pakete wie beispielsweise Java installiert wurden, ist es einsatzbereit.

Neben dem Raspberry Pi benötigen wir selbstverständlich Sensoren um die Daten in der Luft zu messen. Der in dieser Arbeit verwendete Sensor *SDS011* kommt von der Firma *Nova Fitness Co., Ltd.* und ermöglicht die Messung von PM10 und PM2.5 Feinstaubwerten, zu sehen in Abbildung 5.3. Zur Ermittlung der Werte wird die Luft über ein Saugrohr in eine Kammer gezogen, wo sie dann mit einem Laser beschossen wird. Die Größe der Feinstaubpartikel wird dann anhand der Streuung des zurück geworfenen Lichts ermittelt.

Der große Vorteil dieses Sensors ist, dass ein USB-Adapter mitgeliefert wird, welcher den Anschluss an den Raspberry Pi deutlich erleichtert. Sensoren für andere Schadstoffwerte, welche in dieser Umsetzung nicht verwendet wurden, müssen über die GPIO-Pins des Raspberry Pi's angeschlossen werden.
Zur Standorterkennung kommt ein handelsüblicher GNSS-Empfänger zum Einsatz, welcher ebenfalls per USB and den Raspberry Pi angeschlossen werden kann.

Abbildung 5.3: Nova Fitness SDS011 Feinstaubsensor

Software

Im folgenden gehen wir auf die Implementierung der Anwendung auf dem Raspberry Pi ein. Abbildung 5.4 zeigt ihre Systemarchitektur. Zum Ansprechen der Sensoren werden Python-Skripts verwendet, welche das Auslesen der Daten im Vergleich zu Java deutlich einfacher gestalten. Zur Datenübermittlung verwenden die Python-Skripts Sockets, über die unsere Siddhianwendung bei erfolgreicher Verbindung die empfangenen Sensordaten an den entsprechenden Sensorhandler übermittelt. Diese leiten die in ein Ereignisformat umgewandelten Daten mithilfe spezifischer Inputhandler an die jeweiligen Eingangsströme in unserer SiddhiEventProcessing-Einheit. Dort werden die aus dem Konzept bereits bekannten Crowdworker-EPAs erzeugt, die für die Ereignisverabeitung verantwortlich sind. In ihnen ist ebenfalls die Verarbeitungslogik hinterlegt, welche der SiddhiManager benötigt, um die SiddhiAppRuntime zu erzeugen. Deren Callback-Methode benutzen wir letztlich, um die Vorverarbeiteten Daten an den Server zu senden.

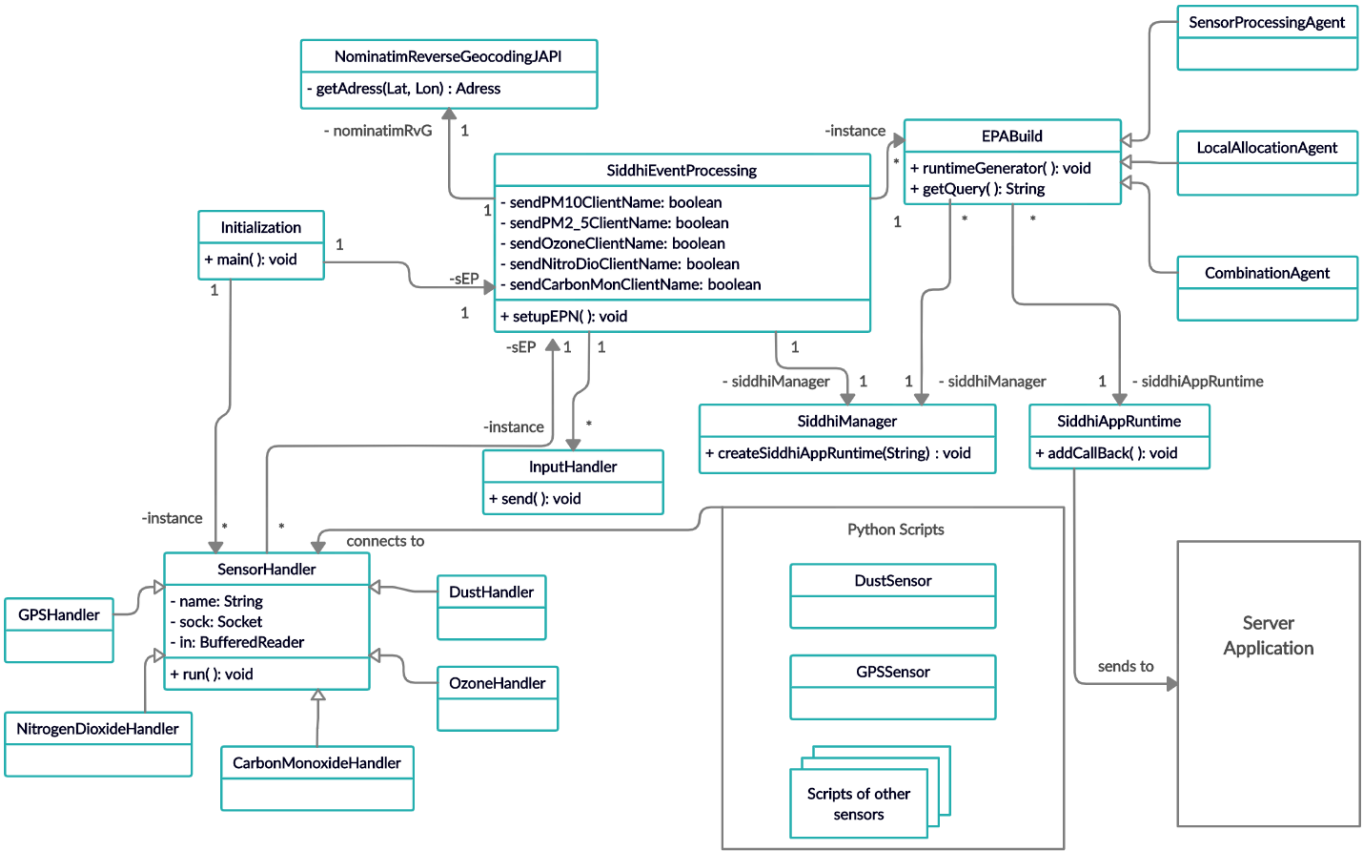

Abbildung 5.4: Systemarchitektur-Diagramm der Crowdworkeranwendung

Sensorverarbeitung

Zur Verarbeitung der Sensordaten verwenden wir wie bereits erwähnt die Programmiersprache Python. Es existieren zahlreiche Bibliotheken, um die verschiedene Datenformate der Sensoren auszulesen. Im Netz existieren außerdem bereits eine Vielzahl von Skripts zur Sensorsteureung, die auch ohne besondere Vorkenntnisse in Python an die spezifischen Anforderungen angepassten und verwendet werden können.

Zum auslesen der GPS-Daten bietet sich der Dienst *GPSD* an. Dabei handelt es sich um eine Anwendung, die eine Verbindung zu unserem über USB verbundenen GNNS-Sensor herstellt, die eintreffenden Daten erfasst und in verwertbare Geschwindigkeits- und Positionsinformationen umwandelt. Diese Daten können dann über einen TCP-Port auf dem lokalen Raspberry Pi von anderen Anwendungen zur weiteren Verarbeitung abgefragt werden. [Bac17] Ohne den GPSD-Dienst würden wir lediglich für uns Menschen nicht interpretierbare Rohdaten erhalten, wie in Abbildung 5.5 zu sehen. Dazu muss der Dienst jedoch erst installiert und eingerichtet werden. Dies beinhaltet unter anderem die Zuweisung des USB-Ports und einen automatischen Start des GPSD-Dienstes bei jedem Start des Raspberry Pis. Die Datenabfrage der GPSD-Daten übernimmt ein Python-

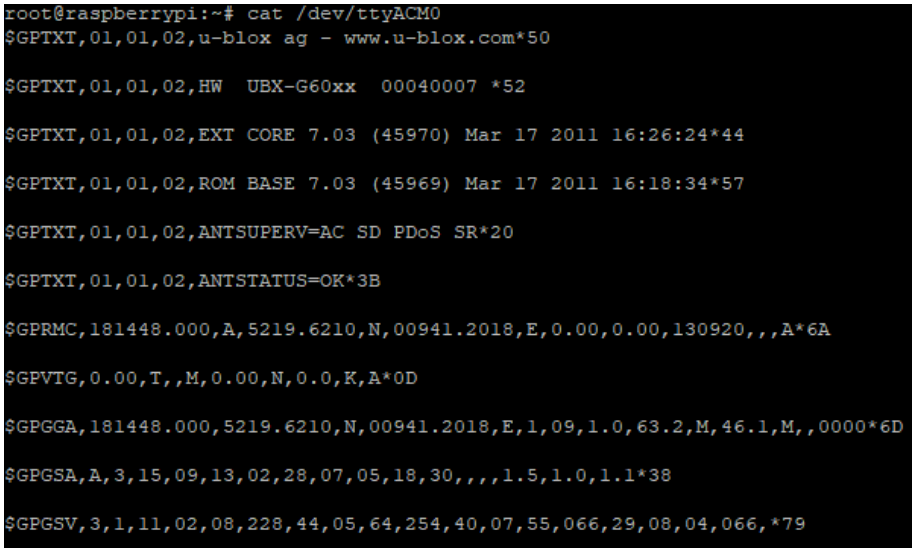

Abbildung 5.5: Rohdaten des GNNS-Empfängers

Skript. Dieses beschafft sich die Daten mit einer dafür bereitgestellt Bibliothek[1] und sendet die Daten per Socket an die lokale Siddhianwendung des Raspberry Pis, zu sehen in Abbildung 5.6. Dazu wird zunächst versucht, einem Verbindung zu dem Dienst herzustellen. Wenn dies funktioniert, wird eine Verbindung zur Siddhianwendung hergestellt und der Sensorname zur Identifikation versendet. Bei erfolgreicher Verbindung werden kontinuierlich die aktuellen Daten des GPSD-Dienstes abgefragt und nacheinander an unsere Anwendung versendet. Voraussetzung dafür ist jedoch ein 2D-Fix. Dies bedeutet, dass der Sensor sich mit mindestens zwei Sateliten verbunden haben muss, um mittels Triliteration[2] die Positionsbestimmung durchzuführen. Damit sind die Vorkehrungen zum Auslesen der GPS-Daten abgeschlossen, als nächstes folgt die Kommunikation mit dem Feinstaubsensor.

[1] Weitere Informationen zur verwendeten Bibliothek unter https://github.com/MartijnBraam/gpsd-py3

[2] Entfernungsmessung anhand von drei Punkten

```python
import gpsd
import sys
import time
import socket

host = "localhost"
port = 4999

def setup():
    try:
        # Connect to the local gpsd
        gpsd.connect()
    except ConnectionRefusedError:
        print("GPSD connection refused")
        print("Maybe the service didn\'t start properly")
        sys.exit()

def loop():
    while True:
        try:
            packet = gpsd.get_current()
            data = str(packet.hspeed) + "\n"
            sock.sendall(data.encode())
            data = str(packet.lat) + "\n"
            sock.sendall(data.encode())
            data = str(packet.lon) + "\n"
            sock.sendall(data.encode())
            time.sleep(5)
        except gpsd.NoFixError:
            print("Needs at least a 2D fix")
            destroy()
            sys.exit()

def destroy():
    sock.close()

if __name__ == '__main__':
    setup()
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((host, port))
        sock.sendall(b"GpsSensor\n")
        loop()
    except KeyboardInterrupt:
        destroy()
```

Abbildung 5.6: Python-Skript zum Auslesen der GPS-Daten

Der Feinstaubsensor liefert uns zunächst Daten im hexadezimalen Format (Abbildung 5.7). Aus dem Grund ist es nötig, die Daten mit aufwendigen Rechnungen in für uns verwertbare Messwerte umzuwandeln. Dazu kann das Datenblatt des Sensors konsultiert werden, um zu entnehmen wie die Formel zur Umrechnung definiert ist:

PM2.5 value: PM2.5 ($\mu g/m^3$) = ((PM2.5 High byte * 256) + PM2.5low byte)/10

PM10 value: PM10 ($\mu g/m^3$) = ((PM10 high byte * 256) + PM10 low byte)/10

```
0000000 aac0 8c00 9400 bcf9 d5ab
```

Abbildung 5.7: Rohdaten des Feinstaubsensors

The number of bytes	Name	Content
0	Message header	AA
1	Commander No.	C0
2	DATA 1	PM2.5 Low byte
3	DATA 2	PM2.5 High byte
4	DATA 3	PM10 Low byte
5	DATA 4	PM10 High byte
6	DATA 5	ID byte 1
7	DATA 6	ID byte 2
8	Check-sum	Check-sum
9	Message tail	AB

Abbildung 5.8: Kommunikationsprotokoll des Sensors, vgl. [NFC15]

Demnach müssen jeweils die Dezimalwerte des high byte mit 256 multipliziert werden. Anschließend wird der Dezimalwert des low byte darauf addiert und das Gesamtergebnis durch 10 dividiert. Das Resultat ergibt den jeweiligen gemessenen Feinstaubwert in Mikrogramm pro Kubikmeter Luft.

Die Umrechnung der Werte in Python muss dabei in diesem Fall nicht zwingend selbst implementiert werden, da bereits zahlreiche Skripts vorhanden sind, die diese Berechnungen implementieren. Dennoch ist es von Vorteil nachvollziehen zu können, in welcher Form der Sensor die Daten übermittelt und wie sie ausgelesen werden können, da andere Sensoren ähnlich funktionieren, aber ggf. keine einsatzfähigen Python-Anwendung angeboten werden. Das in dieser Arbeit verwendete Skript[3] wurde entsprechend angepasst, um die Daten an die lokale Siddhianwendung zu senden. Abbildung 5.9 zeigt den Teil des Skripts, welcher für den Versand der gemessenen Daten zuständig ist. Wie bei den GPS-Daten wird zunächst versucht, eine Socket-Verbindung zu der Siddhianwendung herzustellen. Daraufhin wird auch hier der Sensorname zur Identifikation mitgeteilt. Die umgerechneten Daten werden in einem Array gespeichert, aus dem die Werte anschließend gelesen und übermittelt werden, sofern das Array nicht leer ist.

[3]Das Skript ist zu finden unter https://github.com/zefanja/aqi/blob/master/python/aqi.py

```python
if __name__ == "__main__":
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((host, port))
    sock.sendall(b"DustSensor\n")
    while True:


        cmd_set_sleep(0)
        cmd_set_mode(1);
        for t in range(15):
            values = cmd_query_data();
            if values is not None:

                pm10, pm2_5 = values[1], values[0]
                data = str(pm10) + "\n"
                sock.sendall(data.encode())
                data = str(pm2_5) + "\n"
                sock.sendall(data.encode())
                time.sleep(2)
```

Abbildung 5.9: Ausschnitt des Python-Skripts zur Sensorsteuerung

Ebenfalls anzumerken ist, dass die Laserdiode des Sensors über eine begrenzte Lebensdauer von ca. 8000 Stunden [NFC15] verfügt. Demnach würde der Sensor im Dauerbetrieb nach knapp einem Jahr nicht mehr funktionsfähig sein. Dies kann hinausgezögert werden, in dem der Sensor in gewissen Abstände in einen Schlafzustand versetzt wird. Diese Funktion ist ebenfalls in dem Skript implementiert und kann je nach Anforderungen angepasst werde.

Die Sensorverarbeitung ist damit abgeschlossen und wir gehen über zu unserer Hauptanwendung.

Siddhi auf dem Raspberry Pi

In Abschnitt 5.1 wurden die wesentlichen Strukturen der CEP-Enginge Siddhi und der grundsätzlichen Aufbau einer Siddhianwendung in Java vorgestellt. Im folgenden werden aus diesem Grund leidiglich die Regeldefinitionen der aus dem Konzept bekannten EPA's beschrieben. Da ihnen eine deutlich große Regelmenge zur Ereignisverarbeitung vorliegt, beschränken wir uns auf die Verarbeitung der eintreffenden Daten des hier verwendeten Feinstaubsensors. Diese Regeln unterscheiden sich in den meisten Fällen nicht besonders zu denen der anderen Schadstoffe. Bei großen Unterschieden wird jedoch ebenfalls auf diese eingegangen.

Sensor Processing Agent

Um die Sensordaten für die Verarbeitung empfangen zu können, definieren wir uns zunächst einen Eingangsstrom pro Sensor, in Abbildung 5.10 zu sehen als *GPSEvent-Stream* und *PM10EventStream*.

```
String gpsStream = "define stream GPSEventStream (symbol string, speed float, lat float, lon float); ";
String pm10Stream = "define stream PM10EventStream (symbol string, msrdValue float);";

String cleanedGpsSink = "@sink(type='inMemory', topic='cleanedGPS', @map(type='passThrough')) "
                        +"define stream CleanedGPSStream (symbol string, speed float, lat float, lon float);";
String cleanedPm10Sink = "@sink(type='inMemory', topic='cleanedPM10', @map(type='passThrough')) "
                         +"define stream CleanedPM10Stream (symbol string, msrdValue float);";
```

Abbildung 5.10: Definition von Eingabeströmen und Sinks des Sensor Processing Agent

Außerdem zu sehen sind die Definitionen der in Unterabschnitt 5.1.2 beschriebenen Sink-Ausgangsströme, zu erkennen an dem *@sink*-Tag. Diese Sinks benötigen eine Art Signatur, in der verschiedene Informationen hinterlegt sein müssen. Das Schlüsselwort *topic* legt den Namen des Publishing-Themas fest, anhand dessen andere Ereignissströme die eintreffenden Ereignisse dieses Ausgabestroms abonnieren können. Die Art der Datenhaltung wird hier durch *type='inMemory'* beschrieben. Dies bedeutet, dass die Daten im Arbeitsspeicher vorgehalten und von dort abgerufen werden. Das Tag *@map(type='passThrough'))* sagt aus, dass die Daten in der Form an die Subscriber-Datenströme weitergeleitet werden sollen, in der sie auch in den Sinks ankommen. Die Definition der jeweiligen Subscriber-Ereignisströme entspricht derjenigen der abonnierten Sinks, mit dem Unterschied des am Anfang stehenden *@source*-Tags. Der Rest der der Signatur muss sich mit Ausnahme des Ereignisstrom-Namens zwingend mit der des Sinks gleichen.

Die Definitionen der verschiedenen Ereignisströme erfolgt analog dem hier beschriebenen Schema, weshalb sie hier im weiteren Verlauf nicht weiter betrachtet werden.

```
String cleanGPSQuery = "@info(name = 'gpscleaning') " +
        "from GPSEventStream[speed < 10]  " +
        "select symbol, speed, lat, lon " +
        "insert into CleanedGPSStream ;";

String cleanPM10Query = "@info(name = 'pm10cleaning') " +
        "from PM10EventStream[(msrdValue >= 0 and msrdValue <= 200)]  " +
        "select symbol, msrdValue " +
        "insert into CleanedPM10Stream ;";
```

Abbildung 5.11: Definition der Verarbeitungsregeln des Sensor Processing Agent

Abbildung 5.11 zeigt die Regeldefinitionen des Sensor Processing Agents zur Datenbereinigung. Für die GPS-Daten betrachten wir ausschließlich Ereignisse, die bei einer Geschwindigkeit von unter 10km/h erfasst wurden. Damit filtern wir zum einen unrealistische, sprunghafte Standortänderungen aus unserem Ereignisstrom. Andererseits gewährleisten wir damit, dass wir keine Daten verwerten, die während der Fortbewegung mit öffentlichen Verkehrsmitteln oder anderen Fortbewegungsmitteln ermittelt wurden (siehe Abschnitt 4.2). Das Vorgehen für die Feinstaub-Ereignisse erfolgt auf ähnliche Weise. Wir filtern dafür alle negativen Ereignisse, sowie Ereignisse mit Werten über $200\mu g/m^3$ aus dem Datenstrom. Die gefilterten Ereignisse werden dann in die jeweiligen vorher definierten Ausgabeströme *CleanedGPSStream* und *CleanedPM10Stream* geleitet. Hier müssen die Signaturen in der Anfrage und dem Sink hinsichtlich der Attribute ebenfalls identisch sein.

Combination Agent

Die bereinigten Daten werden nun vom Combination Agent zusammengefügt, um einen ersten räumlichen Bezug der Feinstaubdaten zu erhalten. Abbildung 5.12 zeigt die dafür verwendete Regel. Diese erwartet ein bereinigtes Feinstaub-Ereignis (*e1=CleanedPM10Stream*).

```
String pm10gpsCombiQuery = "from every( e1=CleanedPM10Stream ) -> e2=CleanedGPSStream "
        + "within 3 sec "
        + "select e1.symbol as symbol, e1.msrdValue as msrdValue, e2.speed as speed, e2.lat as lat, e2.lon as lon "
        + "insert into LocalPM10Stream; ";
```

Abbildung 5.12: Definition der Verarbeitungsregeln des Combination Agent

Innerhalb von drei Sekunden muss darauf ein bereinigtes GPS-Ereignis (*e2=CleanedGPSStream*) eintreten, um die Informationen zusammenzuführen. Realisiert wird dies durch den aus

Kapitel 3 bekannten Sequenzoperator. Die drei Sekunden dienen unter anderem als Puffer, falls der Feinstaubmesswert unter korrekten Voraussetzungen erfasst wurde und der GPS-Sensor einen fehlerhaften Wert im Sinne eines Standortsprungs liefert. Sobald die Regel feuert, werden die Werte beider Ereignisse zusammengeführt und in den *LocalPM10Stream* geleitet.

Local Allocation Agent

Nun müssen die kombinierten Ereignisse mit Kontextwissen angereichert werden, um die GPS-Daten in verständliche Informationen umzuwandeln. Dafür werden mithilfe der Callback-Methode des LocalPM10Streams aus jedem eintreffenden Ereignis zunächst die Latitude- und Longitude-Werte extrahiert. Diese Daten werden unter Zuhilfenahme einer externen Reverse Geocoding Bibliothek[4] in für uns lesbare Adressdaten umgewandelt. Ein Inputhandler sendet diese gewonnen Informationen zusammen mit den zugehörigen Latitude- und Longitudedaten an einen *RevGeocodingStream*, der als Zwischenereignisstrom dient. Das beschriebene Vorgehen ist in Abbildung 5.13 dargestellt. Das Muster für die Zuordnung der Adressdaten zu den Feinstaubereignissen ist in Abbildung 5.14 dargestellt. Dafür muss der LocalPM10Stream mit dem eben beschriebenen Zwischenstream gejoined werden. Beide Ereignisse müssen dabei innerhalb einer Sekunde eintreffen. Die Übereinstimmung der jeweiligen Latitude- und Longitudewerte aus beiden Streams stellt das Join-Kriterium dar. Trifft dies ein, werden die zugehörigen Adressdaten zusammen mit den Schadstoffmesswerten in einem neuen Ereignis dem *PM10StreetStream* hinzugefügt.

```
laa.siddhiAppRuntime.addCallback( streamId: "LocalPM10Stream", (StreamCallback) (events) -> {
        for (Event e: events){
            String latStr = e.getData()[3].toString();
            String lonStr = e.getData()[4].toString();

          try {
              streetInput.send(new Object[]{nominatim2.getAdress(Double.parseDouble(latStr),Double.parseDouble(lonStr)),
                                                                  Double.parseDouble(latStr),Double.parseDouble(lonStr)});
          } catch (InterruptedException interruptedException) {
              interruptedException.printStackTrace();
          }
        }

});
```

Abbildung 5.13: Anreicherung mit Kontextwissen durch den Local Allocation Agent

```
String pm10streetQuery = "from LocalPM10Stream#window.time(1 sec) as L "
        + "join RevGeocodingServiceStream#window.time(1 sec) as R "
        + "on L.lat == R.lat and L.lon == R.lon "
        + "select L.symbol as symbol, L.msrdValue as msrdValue, R.street as street "
        + "insert into PM10StreetStream; ";
```

Abbildung 5.14: Definition der Verarbeitungsregel des Local Allocation Agent

Die Verarbeitung auf dem Raspberry Pi ist damit abgeschlossen und die Daten müssen

[4]Weiter Informationen zur verwendeten Reverse Geocoding Bibliothek unter https://www.daniel-braun.com/technik/reverse-geocoding-library-for-java/: :text=I've%20written%20a%20small,address%5D%20service%20based%20on%20OpenStreetMap.

nun an den Server gesendet werden. Dafür kommt die Callback-Methode des PM10StreetStreams zum Tragen. Das Vorhaben zeigt Abbildung 5.15. Zunächst wird ein Thread definiert, der für den Versand der eintreffenden Ereignisse zuständig ist. Er versucht eine Verbindung zum Server aufzubauen und teilt bei Erfolg mit, welche Daten gesendet werden. Anschließend werden die relevanten Daten aus den kontinuierlich eintreffenden Ereignissen extrahiert und an den Server gesendet.

```
Thread t1 = new Thread( () -> {
    try {
        Socket socket1 = new Socket( host: "IP-ADDRESS", port: 5000);
        BufferedWriter bw1 = new BufferedWriter(new OutputStreamWriter(socket1.getOutputStream()));
        laa.siddhiAppRuntime.addCallback( streamId: "PM10StreetStream", (StreamCallback) (events) -> {
                EventPrinter.print(events);
                try {
                    if(!sendPM10ClientName) {
                        bw1.write( str: "PM10Data" + "\n");
                        bw1.flush();
                        sendPM10ClientName=true;
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
                for (Event e : events) {
                    try {
                        bw1.write( str: e.getData()[1].toString()+"\n");
                        bw1.write( str: e.getData()[2].toString()+"\n");
                        bw1.flush();
                    } catch (IOException ioException) {
                        ioException.printStackTrace();
                    }
                }
        });
    } catch (IOException e) {
        e.printStackTrace();
    }
} );
```

Abbildung 5.15: Übermittlung der angereicherte Ereignisse an den Server durch die Callback-Methode des PM10StreetStream

5.2.2 Der Server

Die Serveranwendung ist im Prinzip so aufgebaut wie die Anwendung auf dem Raspberry Pi. Die Systemarchitektur ist Abbildung 5.16 zu entnehmen. Zur Koordinierung der Crowdworker-Clients bedient sich die Initialization je Schadstoff einem Executor mit einem Threadpool, um die verschiedenen Anfragen zeitnah zu bearbeiten. Je nach Art der erhobenen Daten werden diese durch den Executor an einen entsprechende Coordinator weitergeleitet. Dieser generiert aus ihnen Ereignisse und sendet sie über die jeweiligen InputHandler an die verschiedenen Eingangströme in der CentralSiddhiEventProcessing-Komponente zur Verarbeitung durch die Server-EPAs.

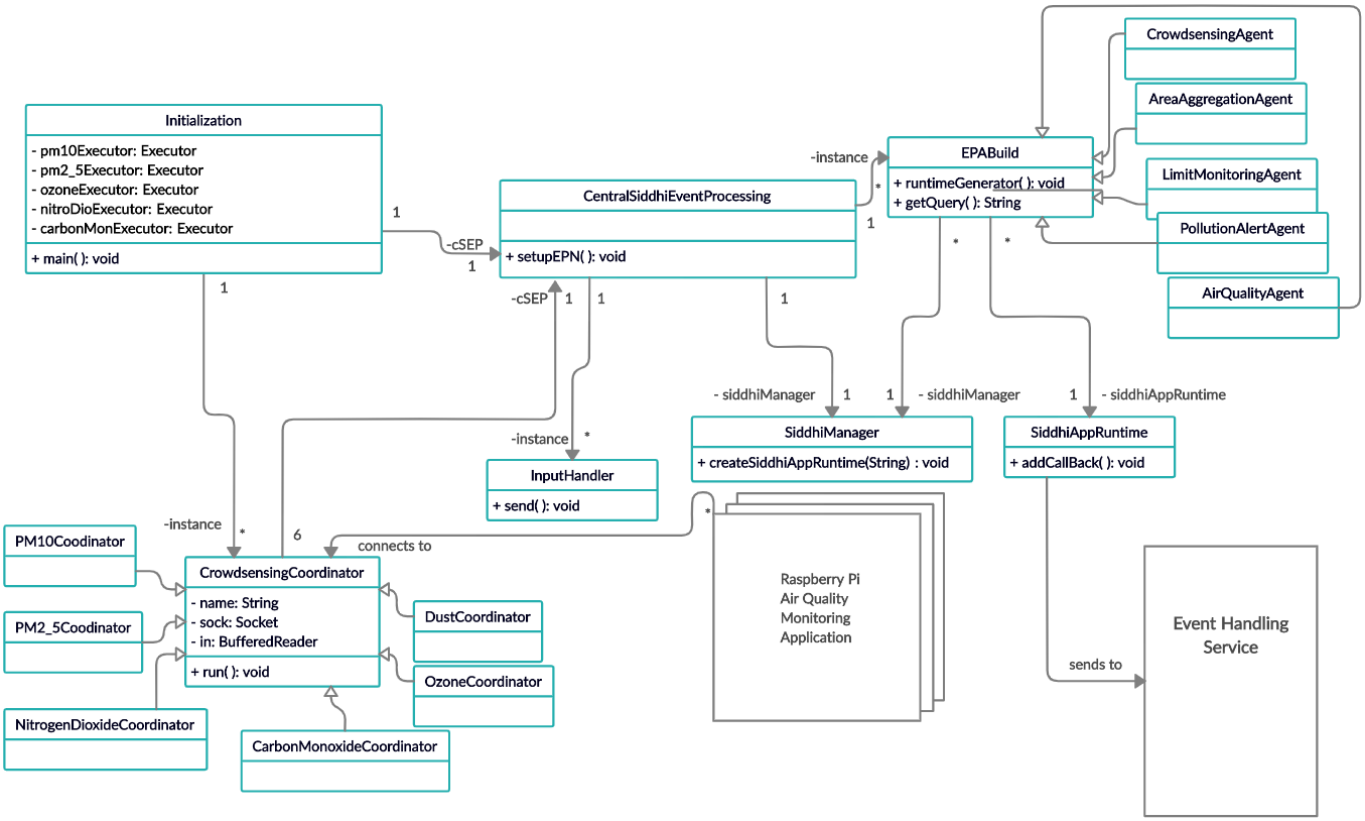

Abbildung 5.16: Systemarchitektur-Diagramm der Serveranwendung

Siddhi auf dem Server

Wie auch auf dem Raspberry Pi werden im folgenden lediglich die Regeln für die Feinstaubereignisse und bei nennenswerten Unterschieden auch die der anderen Schadstoffe vorgestellt. Die Definition der verschiedenen Ereignisströme erfolgt ebenfalls wie in Abschnitt 5.2.1 beschrieben und wird hier nicht weiter behandelt.

Crowdsensing Agent

Wie im Konzept bereits beschrieben, werden die eintreffenden Daten zur Ereignisreduktion zunächst zusammengefasst. Dazu sind drei Schritte nötig, die in Abbildung 5.17 dargestellt sind. In der ersten *summarizedPM10Query* betrachten wir für den PM10-

```
String summarizingPM10Query = "from PM10StreetEventStream#window.timeBatch(1 hour) "
        + "select symbol, msrdValue, count(msrdValue) as amount, street "
        + "group by msrdValue, street "
        + "insert into PM10CountStream; ";
String PM10StreetSepQuery = "from PM10CountStream#window.timeBatch(1 millisecond) "
        + "select symbol, (amount * msrdValue) as multValue, amount, street "
        + "group by msrdValue, street "
        + "insert into PM10StreetSeparationStream; ";
String PM10SumQuery = "from PM10StreetSeparationStream#window.timeBatch(1 millisecond) "
        + "select symbol, (sum(multValue) / sum(amount)) as avgVal, sum(amount) as sumAm, street "
        + "group by street "
        + "insert into SummarizedPM10Stream; ";
```

Abbildung 5.17: Ereignisreduktion durch den Crowdsensing Agent

Feinstaubwert die eintreffenden Ereignisse innerhalb einer Stunde, wozu wir uns ein timeBatch-Window definieren. Um die Häufigkeiten der einzelnen Messwerte pro Straße zu detektieren, wenden wir die Aggregierungsfunktion *count* auf die Messwerte an und Gruppieren diese anhand dieser Messwerte und der Straße. Die Ergenisse leiten wir in einen Zwischenstrom names *PM10CountStream*, welcher den Wert der einzelnen Häufigkeitsereignisse berechnet, in dem die Messwerte mit der Anzahl ihres Vorkommens pro Straße multipliziert werden. Hier ist erneut eine Gruppierung anhand der Messwerte und der Straße nötig. Aus diesem Strom leiten wir dann unter anderem den berechneten Wert, weiterhin die Anzahl der Häufigkeiten und den zugehörigen Straßennamen in einen letzten *PM10StreetSeparationStream*. Diese Regelun den Durchschnitt aus den erfassten Werten in dem festgelegtem Zeitraum des ersten Eingangsstroms (PM10StreetStream), in dem die berechneten Werte sowie die ermittelten Häufigkeiten aufsummiert und miteinander dividiert werden. Um die Durchschnitte pro Straße zu erhalten, Gruppieren wir diese anhand des Straßen-Attributs. Somit erhalten wir eine reduzierte Anzahl an Ereignissen, die weiterhin die vollständigen Informationen der eingangs eintreffenden Datenmenge enthält. Diese voraggregierten Ereignisse werden nun an die nächste Verarbeitungsstufe weitergeleitet.

Area Aggregation Agent

Nun müssen die verschiedenen Durchschnittswerte zum einen gemäß ihres Betrachtungszeitraums bezogen auf den Luftqualitätsindex berechnet werden. Für den PM10-Feinstaubwert bedeutet dies nach Abbildung 4.2 aus dem Konzeptionskapitel, dass für diesen ein stündlich gleitender Tagesmittelwert berechnet wird. Da dem keine Berechnungsformel vorliegt, erfolgt die Regeldefinition nach eigener Interpretation. Demnach wird zunächst der Tagesmittelwert aus den letzten 24 Stundenmittelwerten berechnet.

Für diesen wird dann für jeden neu eintreffenden Stundendurchschnitt für die letzten 24 Werte eine erneute Berechnung durchgeführt, so dass sich ein stündlich gleitender Tagesdurchschnitt ergibt. Die Regelumsetzung zeigt Abbildung 5.18. Für die gleitende

```
String avgPM10Query = "partition with ( street of SummarizedPM10Stream ) " +
        "begin "+
        "from SummarizedPM10Stream#window.length(24) "
        + "select avg(avgVal) as fullAvg, street "
        + "insert into FloatingAvgPM10Stream; "
        + "end;";
```

Abbildung 5.18: Stündlich gleitender PM10-Durchschnitt durch den Area Aggregation Agent

Berechnung müssen wir eine Partitionierung einführen. Diese schafft voneinander isolierte Partitionsinstanzen, in welchen jeweils alle Ereignisse mit einem übereinstimmenden Partitionsschlüssel vereint werden. Diese Instanzen können so parallel mit der gleichen Anfragen verarbeitet werden, ohne für jede Instanz eine spezifische Anfrage formulieren zu müssen. Den Partitionsschlüssel stellt hier das Straßen-Attribut dar. Damit berechnen wir für ein Window der Länge 24 anhand der voraggregierten Stundendurchschnitte den gleitenden Tagesdurchschnitt pro Straße. Damit dieser stündlich gleitet, darf kein Batch-Window verwendet werden. Das Ergebnis leiten wir dann in einen *FloatingAvgPM10Stream*.

Für die anderen Schadstoffe werden gewöhnliche Durchschnitte gemäß ihres spezifischen Betrachtungszeitraums berechnet. Abbildung 5.19 zeigt dies beispielhaft anhand des PM2.5-Feinstaubs. Diese Berechnung stellt erneut eine Voraggregierung für eine

```
String avgPM2_5Query = "from SummarizedPM2_5Stream#window.timeBatch("+LocalDate.now().getMonth().length( leapYear: LocalDate.now()
        .getYear() % 4 == 0 && (LocalDate.now().getYear() % 100 == 0 && LocalDate.now().getYear() % 400 == 0))+" days)[ avgVal > 0.0 and sumAm > 0] "
        + "select avg(avgVal) as fullAvg, street "
        + "group by street "
        + "insert into MonthlyAvgPM2_5Stream; ";
```

Abbildung 5.19: Monatlicher PM2.5-Durchschnitt durch den Area Aggregation Agent

nachfolgende Durchschnittsberechnung dar. Das Besondere hierbei ist, dass diese Regel den monatlichen Durchschnitt berechnet. Die Siddhi-Dokumentation liefert leider keine Information darüber, wie ein Zeitfenster über einen Monat behandelt wird. Es könnte sein, dass Siddhi für jeden Monat beispielsweise 30 Tage betrachtet, was zu einer Überschneidung mit anderen Monaten führt und die Daten ihren zeitlichen Bezug nicht konsistent halten können. Um die korrekte Anzahl an Tagen in dem aktuellen Monat zu erhalten, werden hier deshalb Informationen über den aktuell laufenden Monat eingeholt. Zusätzlich wird geprüft, ob das aktuelle Jahr ein Schaltjahr ist. Das berechnete Ergebis wird dann in einen *MonthlyAvgPM25Stream* geleitet.

Von hier an teilen Sich die Ereignisströme unterschiedlich auf drei verschiedenen EPA's auf, wie auch Abbildung 3.2 aus Kapitel 4 zu entnehmen ist.

Limit Monitoring Agent

Einer davon ist der Limit Monitoring Agent, welcher die Ergebnisse zur generellen Grenzwertüberwachung geeignet aggregiert. Dies geschieht erneut über Durchschnittsberechnungen, weshalb eine Darstellung seiner Regeln an dieser Stelle nicht weiter erforderlich ist.

Pollution Alert Event

Um nun zu erkennen, ob die festgelegten Richtlinien zur Einhaltung der Grenzwerte befolgt wurden, ist es nötig zu ermitteln wie oft ein zulässiger Grenzwert innerhalb seines jeweiligen Betrachtungszeitraums überschritten wurde. Zur Veranschaulichung wird hier der Ozonwert herangezogen. Dieser darf nach den Kriterien in Abschnitt 4.3 einen 8-Stunden-Durchschnitt von 120μg/m^3 eines Tages nicht öfter als 25 mal im Jahr überschreiten. Dieses Muster ist in Abbildung 5.20 dargestellt: Die Regel erfasst über ein

```
String OzoneYearLimitCountQuery = "from EightHourAvgOzoneStream#window.timeBatch(1 year)[ eightHourLimit > 120 ] " +
        "select count(eightHourLimit) as countLimit, street " +
        "group by street " +
        "having countLimit >25 " +
        "insert into OzoneYearLimitCountAlertStream;";
```

Abbildung 5.20: Grenzwertüberwachung durch den Pollution Alert Agent

Jahr verteilt alle Werte, die über der zulässigen Grenze liegen. Die Anzahl dieser erhöhten Werte wird mit der count-Funktion in einem neuen Attribut *countLimit* gespeichert. Um wieder einen räumlichen Bezug zu erhalten, werden die Werte anhand ihrer Straße mithilfe der Gruppierungs-Funktion zusammengefasst. Als letztes definiert eine *Having*-Klausel die Bedingung, welche entscheidet ob die Richtlinien eingehalten wurden. Genau genommen prüft sie nämlich, ob die Anzahl der Überschreitungen noch im zulässigen Rahmen liegt oder nicht.

Den Kriterien aus Abschnitt 4.3 ist ebenfalls zu entnehmen. dass bei bestimmten Ausprägungen des Ozonwerts ein besonderes Vorgehen nötig ist. Dazu werden nachfolgend zwei Regeln betrachtet: Sobald ein Ozonwert zwischen 181 und 240 $\mu g/m^3$ erfasst wurde,

```
String OzoneInformationQuery = "from every OneHourAvgOzoneStream[fullAvg < 241 and fullAvg > 180] "
        + "select fullAvg, 'population must be informed' as info, street "
        + "insert into OzoneInformationStream; "
        + ""
        + "from every OneHourAvgOzoneStream[fullAvg > 240] "
        + "select fullAvg, 'Recommendation of behavior to population necessary' as info, street "
        + "insert into OzoneInformationStream;";
```

Abbildung 5.21: Informationsereignisse zur Verhaltensempfehlung und Informationsplficht für Behörden durch den Pollution Alert Agent

sind die örtlichen Behörden dazu verpflichtet die Bevölkerung über diesen Zustand zu unterrichten. Dies ist in der ersten Regel dargestellt. Dem Ereignis wird in der insert-Anweisung quasi eine Notiz hinzugefügt, die diese Information erhält. Bei Werten über 240 $\mu g/m^3$ muss gar eine Verhaltensempfehlung an die Bevolkerung ausgesprochen werden. Dieses Muster wird in der zweiten Regel dargestellt.

Damit ist die Grenzwertüberwachung der Schadstoffee durch die Anwendung beendet und mit der Callback-Methode des *OzoneInformationStream* können diese Ereignisse nun an nachgelagerte Service-Anwendungen gesendet werden, die für das weitere Event-Handling zuständig sind.

Air Quality Agent

Schließlich fehlt noch die Luftqualitäts-Auswertung durch die Anwendung. Diese gestaltet sich teilweise etwas schwierig und zeigt ein wenig die Grenzen der Siddhi Streaming SQL bei der Anfragedefinition auf. Da sich der LQI aus drei Werten zusammensetzt, sofern Werte über alle drei Schadstoffe vorhanden sind, müssen wir Ereignisse aus drei Ereignisströmen betrachten. Dazu sind erneuts Joins nötig. Das Problem dabei ist jedoch zum einen, dass wir je nach Ausprägung der Werte einen anderen LQI erhalten und gesondert betrachten müssen. Innerhalb einer Join-Anfrage können wir jedoch nicht nach Werten filtern. Andererseits können Joins nicht auf drei verschiedene Ereignisströme innerhalb einer einzigen Anfrage angewendet werden. Aus diesen zwei Gründen muss sich für die benötigten Anfragen anders beholfen werden.

Zunächst werden die Ereignisse der verschiedenen Schadstoffe abhängig ihres Werts in die jeweilige LQI-Kategrorie geleitet. Abbildung 5.22 zeigt wie dies am Beispiel des

```
String OzoneAQIPreFilterQuery = "from every OneHourAvgOzoneStream[fullAvg > 240] "
        + "select fullAvg, 'VERY BAD' as aqi, street "
        + "insert into OzoneVeryBadStream;"
        + ""
        + "from every OneHourAvgOzoneStream[fullAvg < 241 and fullAvg > 180] "
        + "select fullAvg, 'BAD' as aqi, street "
        + "insert into OzoneBadStream; "
        + ""
```

Abbildung 5.22: Umleitung der jeweiligen Schadstoffwerte in die LQI-Kategorie durch den Air Quality Agent

Ozonwerts in der Anwendung umgesetzt ist. Je nach Ausprägung werden die Ereignisse in gesonderte Zwischenstreams gelenkt. Gleichzeitig werden sie mit der Information der zugehörigen LQI-Kategorie angereichert. Dieses Vorgehen erfolgt für die weiteren Kategorien pro Schadstoff analog.

Nach der Definition des LQI bestimmt der Wert mit der höchsten Belastung den LQI. Liegt also beispielsweise der gemessene Ozonwert im *bad*-Bereich und die Werte der anderen Schadstoffe unterhalb diesem in ihrer eigenen Spezifikation, ist der Ozonwert hier ausschlaggebend. Um diese Muster ebenfalls zu erkennen, müssen alle Werte unterhalb eines bestimmten Bereichs ebenfalls detektiert werden. Diese Selektierung zeigt Abbildung 5.23 Damit können wir nun verschiedene Join-Anfragen für die verschiede-

```
+ "from every OneHourAvgOzoneStream[fullAvg < 241] "
+ "select fullAvg, street "
+ "insert into OzoneBelowVBStream; "
+ ""
+ "from every OneHourAvgOzoneStream[fullAvg < 181] "
+ "select fullAvg, street "
+ "insert into OzoneBelowBStream; "
```

Abbildung 5.23: Umleitung der jeweiligen Schadstoffwerte in die LQI-Kategorie durch den Air Quality Agent

nen Kombinationsmöglichkeiten bei der Zusammensetzung des LQI definieren. Da sich dadurch eine enorm große Menge ergibt, wird an dieser Stelle nicht auf alles eingegangen und lediglich beispielhaft demonstriert wie das grundlegende Vorgehen dabei ist.

Folgende Regeln definiert eine Anfrage für den bereits beschriebenen Fall, dass der Ozonwert den LQI in diesem Fall aufgrund seines hohen Werts bestimmt. Wir betrachten alle

```
String OzoneBadAQI = "from OzoneBadStream#window.time(1 sec) as a "
        +"join PM10BelowBStream#window.time(1 sec) as b "
        +"on a.street == b.street "
        + "select a.aqi as aqi, 'Ozone Value: ' as oV, a.fullAvg as OzoneVal, 'PM10 Value: ' as pmV, b.fullAvg as PM10Val, a.street as street "
        + "insert into OzoneBadMidStream; "
        + ""
        + "from OzoneBadMidStream#window.time(1 sec) as a "
        +"join NitroDioBelowBStream#window.time(1 sec) as b "
        +"on a.street == b.street "
        + "select a.aqi as aqi, a.oV as oV, a.OzoneVal as OzoneVal, a.pmV as pmV, a.PM10Val as PM10Val, 'NitrogenDioxide Value: ' as ndV, b.fullAvg as NitroDioVal, "
        + "a.street as street "
        + "insert into BadOzoneAQIStream; ";
```

Abbildung 5.24: Auswertung des LQI durch den Air Quality Agent

eintreffenden Werte des OzoneBadStream innerhalb einer Stunde. Diese Ereignisse joinen wir mit Ereignissen aus dem PM10BelowBStream um auzuschließen, dass kein Wert der gleichen LQI-Kategorie wie der des Ozonwerts angehört. Das Join-Kriterium ist erneute das Straßen-Attribut. Aus den eintreffenden Ereignissen selektieren wir uns dann die jeweiligen Werte, die LQI-Kategorie und die Straße und leiten dieses Ereignis in einen Zwischenstrom. Diese Ereignisse werden schließlich auf gleiche Weise noch mit dem fehlenden Wert für Stickstoffdioxid kombiniert, in dem sie mit dem zugehörigen Stream gejoint werden. Dieser Umweg ermöglicht das joinen von 3 Streams, woraus dann der vollständige Luftqualitätsindex resultieren kann.

6 Inbetriebnahme des Prototyps

Für die Umsetzung des Konzepts und damit der gesamten Anwendung sind eine Vielzahl von Crowdworkern nötig. Diese müssen über verschiedene Zeiträume unterschiedliche Daten erheben. Dies ist jedoch im Rahmen dieser Arbeit nicht möglich, was aber nicht ausschließt, dass die allgemeine Funktionsfähigkeit der Anwendung getestet werden muss. Dazu soll in dem Kapitel kurz gezeigt werden, ob die Komponenten der gesamten Client-Server-Anwendung korrekt funktionieren und das gewünschte Verhalten zeigen. Am Ende soll noch gezeigt werden, wie das Ergebnis einer solchen Air Quality Monitoring Durchführung aussehen könnte.

6.1 Der Prototyp

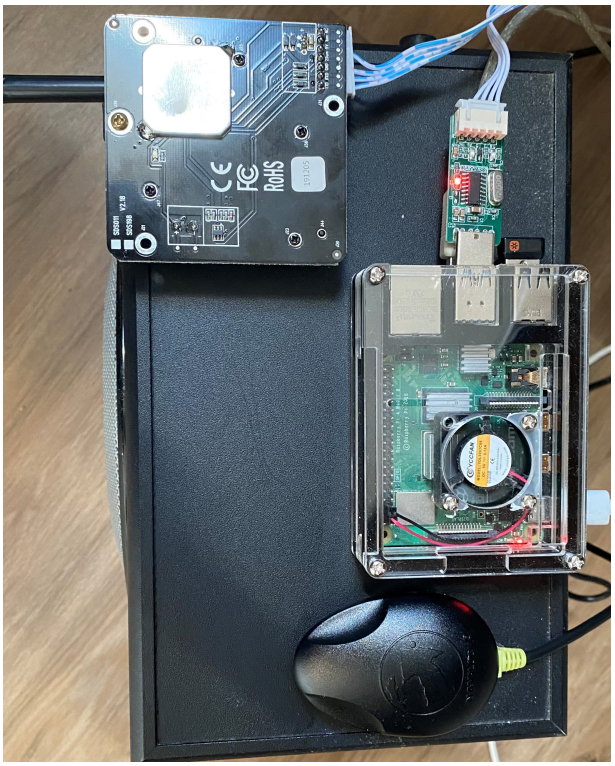

Abbildung 6.1: Verwendeter Prototyp

In Abbildung 6.1 ist der in dieser Arbeit verwendete Prototyp gezeigt. Zu sehen sind das Raspberry Pi mit seinen angeschlossenen Feinstaub- und GNSS-Sensoren. Was deutlich wird, ist dass die Sensoren zum einen Platz benötigen. Schon der Feinstaubsensor hat ähnliche Maße wie das Raspberry Pi selbst. Ein Kohlenmonoxidsensor beispielsweise ist deutlich kleiner, doch lässt sich das nicht pauschal für alle benötigten Sensoren sagen. Außerdem ist es für den Transport erforderlich, alle Komponenten zu gut zu befestigen, damit eine korrekte Funktionsweise der Sensoren gewährleistet ist. Dazu müssen im Falle der tatsächlichen Durchführung des Konzepts weitere Überlegungen gemacht werden, denn dort kommen noch weitere Sensoren hinzu, die einen Transport der Gesamtkonstruktion ohne ausreichende Befestigung deutlich erschweren würden.

6.2 Zusammenspiel der Komponenten

Um die Funktion der Komponenten zu testen, wurden die in den Regeln definierten Zeitfenster entsprechend angepasst. Statt einer Stunden wurde so beispielsweise eintreffende Werte innerhalb der letzten fünf Minuten betrachtet.

Ergebnisse auf dem Raspberry Pi

```
data=[PM10, 12.0, Campus Linden, 118, Ricklinger Stadtweg, Ricklingen, Hannover, Region Hannover, Niedersachsen, 30459, Deutschland]
data=[PM2.5, 5.0, Campus Linden, 118, Ricklinger Stadtweg, Ricklingen, Hannover, Region Hannover, Niedersachsen, 30459, Deutschland]
data=[PM10, 12.0, Campus Linden, 118, Ricklinger Stadtweg, Ricklingen, Hannover, Region Hannover, Niedersachsen, 30459, Deutschland]
data=[PM2.5, 7.0, Campus Linden, 118, Ricklinger Stadtweg, Ricklingen, Hannover, Region Hannover, Niedersachsen, 30459, Deutschland]
data=[PM10, 12.0, Campus Linden, 118, Ricklinger Stadtweg, Ricklingen, Hannover, Region Hannover, Niedersachsen, 30459, Deutschland]
data=[PM2.5, 7.0, Campus Linden, 118, Ricklinger Stadtweg, Ricklingen, Hannover, Region Hannover, Niedersachsen, 30459, Deutschland]
```

Abbildung 6.2: Verarbeitete Endergebnisse auf dem Raspberry Pi

Der Test auf dem Raspberry Pi zeigt, dass die einzelnen Komponenten wie erwartet miteinander agieren. Die durch den Feinstaub-Sensor gemessenen Werte werden erfolgreich an die Siddhi-Anwendung gesendet und dort entsprechend verarbeitet. Somit kann die Anwendung von Crowdworkern genutzt werden, um Luftdaten zu sammeln. Abbildung 6.2 zeigt die durch den Prototyp ermittelten Ereignisse, die an den Server gesendet werden. Sie zeigen verschiedene Messdaten für die zwei Feinstaub-Werte auf dem Campus der Hochschule Hannover.

Ergebnisse auf dem Server

Da lediglich ein einziger Prototyp in dieser Arbeit Anwendung findet und keine Crowdworker im Einsatz sind, die Daten für den Server generieren können, kommen Dummy-Clients zum Einsatz. Diese produzieren kontinuierlich Testdaten in gleicher Form wie in Abbildung 6.2, um die Funktion der Server-Anwendung testen zu können. Die Testdaten

bewegen sich dabei jedoch nicht in einem Rahmen, der auch auf die Realität abgebildet werden kann und sollen nur die reine Funktionsfähigkeit des Servers präsentieren.

Auch hier verläuft die Auswertung der Daten wie erwartet. Die verschiedenen Datenströme werden nach den definierten Kriterien koordiniert, so dass am Ende für Menschen verwertbare Informationen vorliegen. Ein mögliches durch den Server produziertes Endereignis zeigt Abbildung 6.3. Diesem fiktivem Ereignis ist zu entnehmen, dass die

Abbildung 6.3: Verarbeitete Endergebnisse auf dem Server

Luftqualität auf dem Campus der Hochschule Hannover mit einem schlechten Index bewertet wurde, welcher sich durch den hohen Ozonwert ergeben hat. Außerdem liegen dem Ereignis zusätzlich Informationen zu den anderen LQI-Bestandteilen vor.

Da die Messwerte immer die aktuelle Live-Situation zu einer genauen Adresse zeigen, könnte nun mithilfe eines Kartendienstes diese Live-Situation auf einer Sensingmap visualisiert werden. Neben der adressgenauen Zuordnung der Auswertungen wäre es auch möglich, über das zu untersuchende Gebiet ein Raster zu legen. Durch dieses Raster würde ein Kartenabschnitt in verschiedenen Bereich eingeteilt werden. Die GPS-Daten müssten dann in Bildkoordinaten überführt werden, wodurch sie sich einem bestimmten Rasterabschnitt zuordnen lassen würden. CEP-technisch gesehen würde dies keinen großen Aufwand erfordern, da sich dadurch lediglich eine minimale Änderung bei der Art der Datengruppierung innerhalb der Anfrageregeln ergeben würde, die sich jedoch ohnehin einfach anpassen lassen. Die Gruppierung geschieht dann nicht mehr anhand der Adressdaten, sondern durch die verschiedenen Rasterabschnitte. Dieses Vorgehen bietet sich jedoch eher an, wenn ein bestimmtes Gebiet festgelegt wird, dass untersucht werden soll. Nur so ist es nämlich möglich, das Gebiet in Abschnitte einzuteilen. Aus diesem Grund wurde in dieser Arbeit auch der Adressansatz verfolgt, da dieser eine ortsunabhängigere Untersuchung ermöglicht.

In dieser Arbeit wurde kein Kartendienst verwendet, um die Daten noch zu visualisieren. Wie das Ergebnis jedoch aussehen könnte, ist in Abbildung 6.4 zu sehen. Diese fiktive

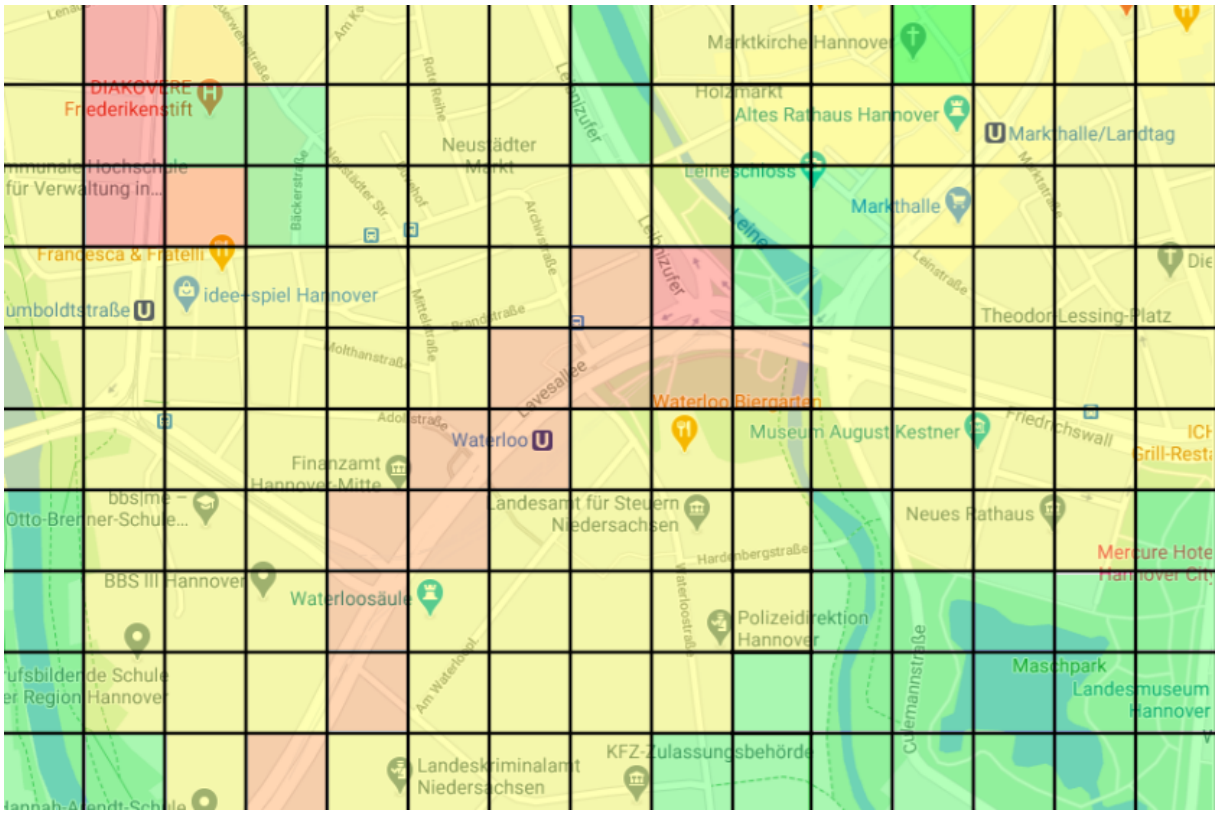

Abbildung 6.4: Mögliche Darstellung einer Sensingmap durch den Server

Sensingmap zeigt, wie sich der LQI über ein bestimmtes Gebiet verteilen kann. So sind beispielsweise Grünflächen und Parks mit einem guten LQI bewertet worden, was durch die grünen Abschnitte auf der Map dargestellt ist (z.B. im unteren rechten Bereich). Viel befahrene Straßen wiederum haben laut der Map teilweise schlechte Bewertungen ergeben, was sich durch die orangenen bis roten Abschnitte erschließen lässt. So können also über jeden Ort eines großflächigen Bereichs Aussagen über die Luftqualität getroffen werden. Zum Vergleich zeigt Abbildung 6.5 die Darstellung eines alleinstehenden, durch eine einzige Messtation ermittelten LQI. Diese Messtation ist durch den Kreis auf dem Bild dargestelt. Aus dieser Darstellung ist nicht zu entnehmen, wie die tatsächliche Luftqualität beispielsweise weiter Rechts in dem Kartenabschnitt ist. Überhaupt ist schwer interpretierbar, für welchen Bereich sich der ausgesprochen Wert zuordnen lässt. In unserer Sensingmap ist das anders, dort sind Aussagen über jeden Ort möglich, da auch an jedem Ort Daten erhoben wurden und die Ergebnisse nicht von einer einzelnen Messtation abgängig sind.

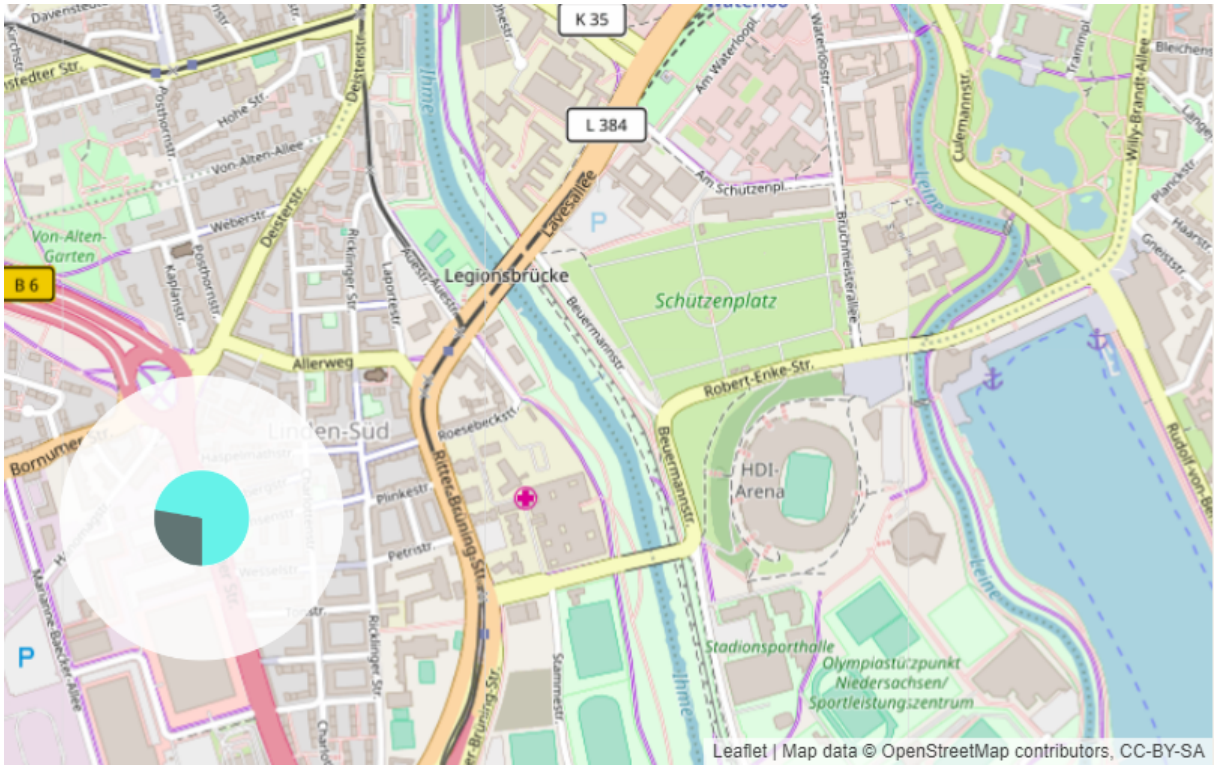

Abbildung 6.5: Darstellung der aktuellen Luftdaten durch das Unweltbundesamt, vgl. [Umw20a]

Neben der Visualisierung können die ermittelten Daten auch direkt an Behörden weitergeleitet werden, die diese Informationen dann nutzen können um Informationen über den aktuellen Zustand und eingehaltene oder auch nicht eingehaltene Grenzwerte der Schadstoffe zu erhalten.

7 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war die Entwicklung eines Konzepts zur crowdbasierten Luftqualitätsmessung. Dafür sollten Raspberry Pis verwendet werden, auf denen die CEP-Technologie zur Datenverarbeitung implementiert ist. Wir haben uns dafür zunächst angeschaut, welcher Gedanke hinter dem Crowdsensing steckt und wie es umgesetzt werden kann. Diese Umsetzung sollte dabei in dieser Arbeit durch Raspberry Pis erfolgen, weshalb die Vorteile diese Geräte und ihre allgemeinen Merkmale und Spezifikationen vorgestellt wurden. Für die Anwendung des CEP auf den Raspberry Pis haben wir uns die Grundlagen dieser Technologie angeschaut und gleichzeitig den dabei verfolgten ereignisbasierten Architekturstil, die Event-Driven Architecture herangezogen. Ausgehend von diesem Fundament wurde dann das angestrebte Konzept entwickelt. Dafür wurden die benötigten Kriterien zur Luftqualitätsmessung detektiert und erforderliche Anforderungen an unsere Crowdworker gestellt. Im Verlauf des Konzepts haben sich dann verschiedenen Ereignistypen ergeben, die durch eine Verarbeitungspipeline verschiedene Schritte zur resultierenden Informationssynthese durchlaufen haben. Diese Verarbeitungsschritte wurde durch unterschiedliche CEP-Agenten repräsentiert, welche im Rahmen des Konzepts einzeln beleuchtet wurden. Anhand dieses Konzepts wurde dann die Anwendung implementiert. Dafür kam die CEP-Engine Siddhi zum Einsatz, welche die Umsetzung des entwickelten Event-Processing-Netzwerks ermöglichte. Die Implementierung umfasste dabei die Crowdworker-Anwendung auf dem Raspberry Pi zur Datenerhebung durch die Sensoren. Auch der Server wurde implementiert, welcher für die Koordination der verschiedenen Crowdworker-Datenströme und ihre schließliche Auswertung zuständig war. Zum Schluss wurde die Funktionsfähigkeit und das Verhalten der gesamten Anwendung getestet und der verwendete Prototyp präsentiert. Schließlich haben wir uns noch ein mögliches Ergebnis des in der Arbeit verfolgten Ansatzes angeschaut und die Unterschiede und Vorteile gegenüber der Ermittlung durch eine konventionelle Messstation dargelegt.

Schon die fiktive Sensingmap hat gezeigt, dass eine Umsetzung des Konzepts deutliche Vorteile gegenüber dem Vorgehen mit einzelnen Messstationen besitzt. Durch die Kombination des Crowdsensing mit der CEP-Technologie können nämlich weitaus präzisere Aussagen über die unterschiedliche Verteilung der Schadstoffe in der Luft gemacht werden. Außerdem konnte das Potential des CEP in diesem Bereich durch die verschiedenen zeitlichen, räumlichen und semantischen Bezüge aufgezeigt werden. Für eine realistische Umsetzung ist es jedoch schwer zu sagen, wie viele Crowdworker letztlich nötig sind, um auch aussagekräftige Ergebnisse zu erhalten. Zum einen hängt die Zahl der Crowdworker davon ab, wie groß das zu untersuchende Gebiet ist. Andererseits spielt auch die

Tatsache mit rein, dass sich die Crowdworker ohne genaue Struktur durch das Gebiet bewegen und so schwer abzuschätzen ist, wie sie sich über das Gebiet während dem Monitoring verteilen werden. Auch die Beschaffung der Geräte und Sensoren bei einer tatsächlichen Umsetzung müsse geklärt werden. Wie werden die Geräte an die freiwilligen verteilt? Wie wird das angesprochene Probleme zum Transport der Geräte gelöst? Wovon das vorgestellte Konzept klar abzugrenzen ist, sind Luftqualitätsmessungen in Innenräumen. Diese unterliegen nämlich anderen Kriterien als Messungen von Außenluft. Auch das crowdsensing würde in diesem Bereich wenig Sinn ergeben. Es können jedoch Überlegungen angestellt werde, wie das Konzept ggf. angepasst werden kann um auch solche Messungen durchführen zu können.

Literaturverzeichnis

[Bac17] Andrew Back. Gps zu raspberry pi-projekten für orts- und zeitbestimmung hinzufügen. 06 2017. Zugegriffen am 23.07.2020.

[BD10] Ralf Bruns and Jürgen Dunkel. *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Xpert.press. Springer, Heidelberg, 2010.

[BD13] Ralf Bruns and Jürgen Dunkel. Towards pattern-based architectures for event processing systems. *Software: Practice and Experience*, 44:1395–1416, 2013.

[BD15] Ralf Bruns and Jürgen Dunkel. *Complex Event Processing - Komplexe Analyse von massiven Datenströmen mit CEP*. Springer Vieweg, 2015.

[Fou20a] Raspberry Pi Foundation. Gpio. https://www.raspberrypi.org/documentation/usage/gpio/, 2020. Zugegriffen am 04.07.2020.

[Fou20b] Raspberry Pi Foundation. Raspberry pi. https://www.elektronik-kompendium.de/sites/com/1904221.htm, 2020. Zugegriffen am 04.07.2020.

[Fou20c] Raspberry Pi Foundation. Raspberry pi 4 tech specs. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/, 2020. Zugegriffen am 09.09.2020.

[Fou20d] Raspberry Pi Foundation. Raspberry pi os (previously called raspbian). https://www.raspberrypi.org/downloads/raspberry-pi-os/, 2020. Zugegriffen am 09.09.2020.

[GYL11] Raghu K. Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, 2011.

[Hed17] Ulrich Hedtstück. *Complex Event Processing: Verarbeitung von Ereignismustern in Datenströmen*. eXamen.press. Springer Vieweg, 2017.

[Inc20a] WSO2 Inc. Siddhi 5.1 architecture. https://siddhi.io/en/v5.1/development/architecture/, 2020. Zugegriffen am 13.07.2020.

[Inc20b] WSO2 Inc. Siddhi 5.1 quick start guide. https://siddhi.io/en/v5.1/docs/quick-start/, 2020. Zugegriffen am 13.07.2020.

[Inc20c] WSO2 Inc. Siddhi 5.1 streaming sql guide. https://siddhi.io/en/v5.1/docs/query-guide/, 2020. Zugegriffen am 13.07.2020.

[Luf] Luftbewusst.de. Luftverschmutzung: Arten, ursachen und folgen. https://luftbewusst.de/umwelt/luftverschmutzung-arten-ursachen-und-folgen/. Zugegriffen am 20.05.2020.

[MMH+12] Xufei Mao, Xin Miao, Yuan He, Xiang-Yang Li, and Yunhao Liu. Citysee: Urban co2 monitoring with sensors. *Proceedings - IEEE INFOCOM*, pages 1611–1619, 03 2012.

[MPR08] Prashanth Mohan, Venkata Padmanabhan, and Ramachandran Ramjee. Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. *Proceedings of the 6th International Conference on Embedded Networked Sensor Systems*, pages 323–336, 01 2008.

[MZY14] Huadong Ma, Dong Zhao, and Peiyan Yuan. Opportunities in mobile crowd sensing. *IEEE Communications Magazine*, 52(8):29–35, 2014.

[NFC15] Ltd. Nova Fitness Co. Laser pm2.5 sensor specification. https://cdn-reichelt.de/documents/datenblatt/X200/SDS011-DATASHEET.pdf, 06 2015.

[Sho10] Cynthia Shollenberger. Cars as traffic sensors. http://news.mit.edu/2010/cars-sensors-0924, 2010. Zugegriffen am 10.06.2020.

[Umw16] Umweltbundesamt. Stickstoffoxide. https://www.umweltbundesamt.de/themen/luft/luftschadstoffe/stickstoffoxide, 2016. Zugegriffen am 02.06.2020.

[Umw19] Umweltbundesamt. Berechnungsgrundlagen luftqualitätsindex. https://www.umweltbundesamt.de/berechnungsgrundlagen-luftqualitaetsindex, 2019. Zugegriffen am 12.06.2020.

[Umw20a] Umweltbundesamt. Aktuelle luftdaten. https://www.umweltbundesamt.de/daten/luft/luftdaten/luftqualitaet/eJzrWJSSuMrIwMhA18BS19ByUUnmIkPzRXmpCy0XFZcstjQ2WpziVoRQAOSG5COrz63iXJSb3LQ4J7HktIOXUeqLuUeLFufkpZ92UDo$_r$y9J6S0Are8ldA ==, 2020.

[Umw20b] Umweltbundesamt. Feinstaub. https://www.umweltbundesamt.de/themen/luft/luftschadstoffe/feinstaub, 2020. Zugegriffen am 02.06.2020.

[Umw20c] Umweltbundesamt. Kohlenmonoxid. https://www.umweltbundesamt.de/themen/luft/luftschadstoffe/kohlenmonoxid, 2020. Zugegriffen am 02.06.2020.

[Umw20d] Umweltbundesamt. Ozon. https://www.umweltbundesamt.de/themen/luft/luftschadstoffe/ozon, 2020. Zugegriffen am 02.06.2020.

[UWB13] Regelungen und strategien. https://www.umweltbundesamt.de/themen/luft/regelungen-strategienluftreinhaltestrategien, 2013. Zugegriffen am 12.06.2020.

[UWB19] Luftmessnetz: Wo und wie wird gemessen? https://www.umweltbundesamt.de/themen/luftmessnetz-wo-wie-wird-gemessen, 2019. Zugegriffen am 12.06.2020.

[Vis17] Niruhan Viswarupan. A beginner's guide to siddhi — complex event processor. https://medium.com/@niruhan/a-beginners-guide-to-siddhi-complex-event-processor-efa4bd68a71c, 2017. Zugegriffen am 11.04.2020.

[WHO18] Mortality and burden of disease from ambient air pollution. https://www.who.int/health-topics/air-pollution#tab=tab_1, 2018. Zugegriffen am 12.06.2020.

[Wie] Melanie Wieland. Industrielle revolution und umweltverschmutzung. https://www.planet-wissen.de/natur/umwelt/umweltverschmutzung/pwieindustriellerevolutionundumweltverschmutzung100.html, year = 2020, note = Zugegriffen am 20.05.2020,.